-	68	((classif$9 near10 packet$1).ti. or (classif$9 near10 packet$1).ab.) and (709/$.ccls. or 370/$.ccls.)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:56
-	31	(((classif$9 near10 packet$1).ti. or (classif$9 near10 packet$1).ab.) and (709/$.ccls. or 370/$.ccls.)) and @ad<19981228	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:33
-	12	6260072.URPN.	USPAT	2004/01/16 09:31
-	37	("4556972" | "5121383" | "5253248" | "5313454" | "5377327" | "5412654" | "5467345" | "5491801" | "5495479" | "5506847" | "5557607" | "5570346" | "5583861" | "5583862" | "5594734" | "5600630" | "5666360" | "5671222" | "5671445" | "5699361" | "5740164" | "5790536" | "5802278" | "5805816" | "5812526" | "5828844" | "5844887" | "5905712" | "5910942" | "5920566" | "5920568" | "5930254" | "5940372" | "5963546" | "5970232" | "5996021" | "6016306").PN.	USPAT	2004/01/16 09:32
-	2	5848233.pn.	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:55
-	23806	filter$3 same day	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:56
-	103	(filter$3 same day) same (classif$9 same filter$1)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:56
-	3	((filter$3 same day) same (classif$9 same filter$1)) and (709/$.ccls. or 370/$.ccls.)	USPAT; EPO; DERWENT; USOCR	2004/01/16 11:04
-	35	tos near3 header	USPAT; EPO; DERWENT; USOCR	2004/01/16 11:10
-	17	(tos near3 header) and classif$8	USPAT; EPO; DERWENT; USOCR	2004/01/16 11:11
-	13	(tos near3 header) and filter$3	USPAT; EPO; DERWENT; USOCR	2004/01/16 11:11
-	9	((tos near3 header) and classif$8) and ((tos near3 header) and filter$3)	USPAT; EPO; DERWENT; USOCR	2004/01/16 11:11
-	19	(("6195697") or ("6385609") or ("6594786") or ("5905715") or ("6446200") or ("5872928") or ("5974237") or ("6259679") or ("6405251")).PN.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/20 14:26
-	3	((("6195697") or ("6385609") or ("6594786") or ("5905715") or ("6446200") or ("5872928") or ("5974237") or ("6259679") or ("6405251")).PN.) and filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/20 14:26
-	4804	(cos or tos or qos) same (packet$1 or frame$1 or datagram$1)	USPAT; EPO; DERWENT; USOCR	2004/01/21 10:21

-	61142	(classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)	USPAT; EPO; DERWENT; USOCR	2004/01/21 10:21
-	366	((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))	USPAT; EPO; DERWENT; USOCR	2004/01/21 10:21
-	2038	((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 10:39
-	22	(((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.) and (((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:07
-	8	("5313455" | "5463620" | "5781532" | "6104700" | "6167027" | "6188698" | "6222844" | "6381649").PN.	USPAT	2004/01/21 11:25
-	330	709/$.ccls. and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:01
-	10750	709/22$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:01
-	143	709/22$.ccls. and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:01
-	17	(709/22$.ccls. and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)) and ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:09
-	6	6412000.URPN.	USPAT	2004/01/21 11:08
-	10	("5251152" | "5495426" | "5838919" | "5870561" | "5903559" | "5923849" | "6028842" | "6046980" | "6137782" | "6209033").PN.	USPAT	2004/01/21 11:08
-	1	((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) and ((709/22$.ccls. and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)) and ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:09
-	27	709/22$.ccls. and (((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:09
-	3	((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) and (("5313455" | "5463620" | "5781532" | "6104700" | "6167027" | "6188698" | "6222844" | "6381649").PN.)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:25
-	3	(((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) and (("5313455" | "5463620" | "5781532" | "6104700" | "6167027" | "6188698" | "6222844" | "6381649").PN.)) and (tos or cos or qos)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:26

-	2	((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)) and ((((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) and (("5313455" | "5463620" | "5781532" | "6104700" | "6167027" | "6188698" | "6222844" | "6381649").PN.)) and (tos or cos or qos))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:27
-	2	((709/22$.ccls. and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)) and ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))) and (tos or qos or cos)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 11:27
-	122693	filter$1 near5 (delet$4 or expir$6 or remov$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:09
-	148	(filter$1 near5 (delet$4 or expir$6 or remov$3)) and 709/22$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:08
-	0	((filter$1 near5 (delet$4 or expir$6 or remov$3)) and 709/22$.ccls.) and (((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:08
-	7766	(filter$1 near5 (delet$4 or expir$6 or remov$3)) and ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:08
-	41	((filter$1 near5 (delet$4 or expir$6 or remov$3)) and ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))) and ((cos or tos or qos) same (packet$1 or frame$1 or datagram$1))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:09
-	754	dynamic adj filter$1	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:09
-	1	(dynamic adj filter$1) and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:09
-	190	(((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))) and (delet$4 or expir$6 or remov$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 12:10
-	16	((((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1))) and (delet$4 or expir$6 or remov$3)) and (((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 13:21
-	2	((((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.) and (((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)))) and (timeout or (time adj out$1))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 13:21

-	22	((((370/229) or (370/230) or (370/230.1) or (370/235)).CCLS.) and (((cos or tos or qos) same (packet$1 or frame$1 or datagram$1)) same ((classif$9 or filter$1) same (packet$1 or frame$1 or datagram$1)))) and (expir$8 or delet$4 or remov$3 or time or (time adj out$1) or timeout$1)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/21 13:22

L Number	Hits	Search Text	DB	Time stamp
-	2	bandwidth adj broker$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 18:52
-	22	bandwidth same broker$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 18:56
-	11936	service near20 level$1	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:05
-	3561	tos or qos	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:07
-	342	(service near20 level$1) and (tos or qos)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:06
-	1	(bandwidth same broker$3) and ((service near20 level$1) and (tos or qos))	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:06
-	2441	tos	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:07
-	4	tos same (service near20 level$1)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:09
-	304	(service near20 level$1) same filter$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:10
-	0	((service near20 level$1) same filter$3) and tos	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:10
-	261	bandwidth$1 and broker$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:10
-	3	((service near20 level$1) same filter$3) and (bandwidth$1 and broker$3)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:11
-	4909	service near3 level$1	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:11
-	9939	admission same control$4	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:12
-	19	(service near3 level$1) same (admission same control$4)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:21
-	9448	ingress and egress	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:21
-	3585	bandwidth adj (allocat$4 or broker$5 or control$4 or request$3)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:22

-	57	(ingress and egress) and (bandwidth adj (allocat$4 or broker$5 or control$4 or request$3))	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:22
-	13	(service near3 level$1) and ((ingress and egress) and (bandwidth adj (allocat$4 or broker$5 or control$4 or request$3)))	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:26
-	168	(admission$1 or access) adj profil$3	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:27
-	1190	((709/226) or (709/225) or (709/229)).CCLS.	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:27
-	7	((admission$1 or access) adj profil$3) and ((("709/226") or ("709/225") or ("709/229")).CCLS.)	USPAT; EPO; DERWENT; USOCR	2001/09/22 19:27
-	59	policy adj server$1	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:04
-	675	713/15$.ccls.	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:04
-	6	(policy adj server$1) and 713/15$.ccls.	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:07
-	1	admission$1 adj profile$1	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:29
-	4276	dynamic$5 near5 filter$3	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:09
-	2	(dynamic$5 near5 filter$3) and (policy adj server$1)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:29
-	306	filter$3 adj criteria$1	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:30
-	39	admission$1 adj10 profile$1	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:29
-	0	(filter$3 adj criteria$1) and (admission$1 adj10 profile$1)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:29
-	17	(dynamic$5 near5 filter$3) and (filter$3 adj criteria$1)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:35
-	47	ingress near4 profil$3	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:35
-	0	(admission$1 adj10 profile$1) and (ingress near4 profil$3)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:35

-	0	(ingress near4 profil$3) and 713/15$.ccls.	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:35
-	3	(ingress near4 profil$3) and filter$3	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:38
-	102511	remov$3 near10 filter$1	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:40
-	84571	remov$3 near5 filter$1	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:42
-	0	(admission$1 adj10 profile$1) and (remov$3 near5 filter$1)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:42
-	2	(policy adj server$1) and (remov$3 near5 filter$1)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:43
-	4	713/15$.ccls. and (remov$3 near5 filter$1)	USPAT; EPO; DERWENT; USOCR	2001/09/25 12:43
-	867	filter$3 near5 (delet$3 or expir$6)	USPAT; EPO; DERWENT; USOCR	2001/09/26 10:48
-	8494	713/$.ccls.	USPAT; EPO; DERWENT; USOCR	2001/09/26 10:48
-	0	(dynamically with (creat$3 or establish$3 or remov$3 or delet$3) with filter$1) and 713/$.ccls.	USPAT; EPO; DERWENT; USOCR	2001/09/26 10:48
-	20	(filter$3 near5 (delet$3 or expir$6)) and 713/$.ccls.	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:24
-	89	dynamically with (creat$3 or establish$3 or remov$3 or delet$3) with filter$1	USPAT; EPO; DERWENT; USOCR	2001/09/26 10:53
-	9	dynamically near3 (creat$3 or establish$3 or remov$3 or delet$3) near3 filter$1	USPAT; EPO; DERWENT; USOCR	2001/09/26 11:45
-	2	("6148336").PN.	USPAT; EPO; DERWENT; USOCR	2001/09/26 11:45
-	1	(("6148336").PN.) and (remov$3 or delet$3 or expir$7)	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:05
-	1	((("6148336").PN.) and (remov$3 or delet$3 or expir$7)) and packet$1	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:06
-	1	((("6148336").PN.) and (remov$3 or delet$3 or expir$7)) and (destination$1 or source$1)	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:09

-	0	(("6148336").PN.) and (admission adj profile$1)	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:10
-	0	(("6148336").PN.) and (admission near4 profile$1)	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:10
-	1	(("6148336").PN.) and polic$3	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:25
-	1	(("6148336").PN.) and (policy or policies)	USPAT; EPO; DERWENT; USOCR	2001/09/26 12:25
-	26	(("5554322") or ("5884033") or ("5953338") or ("5968176") or ("6055571") or ("6130924") or ("6148336") or ("6167451") or ("6167445") or ("6178505") or ("6199113") or ("6256741") or ("6262974") or ("6295527")).PN.	USPAT; EPO; DERWENT; USOCR	2001/09/29 18:39
-	3	("5544322").PN.	USPAT; EPO; DERWENT; USOCR	2001/09/29 18:39
-	4260	relational adj database	USPAT; EPO; DERWENT; USOCR	2002/04/11 13:05
-	11775	707/$.ccls.	USPAT; EPO; DERWENT; USOCR	2002/04/11 13:05
-	1757	(relational adj database) and 707/$.ccls.	USPAT	2002/04/11 13:06
-	296	((relational adj database) and 707/$.ccls.) and api	USPAT	2002/04/11 13:06
-	1778	trigger$3 near5 filter$1	USPAT	2002/04/18 11:26
-	15	(trigger$3 near5 filter$1) and 713/$.ccls.	USPAT	2002/04/18 11:26
-	1	("6178505").PN.	USPAT	2002/04/22 08:29
-	1	(("6178505").PN.) and expir$7	USPAT	2002/04/22 08:29
-	163	tos same filter$3	USPAT	2002/11/06 15:43
-	6	(tos same filter$3) same (type adj2 service$1)	USPAT	2002/11/06 15:45
-	4611	classif$8 same filter$3	USPAT	2002/11/06 15:45
-	5	(classif$8 same filter$3) same tos	USPAT	2002/11/06 15:46
-	14	(classif$8 same filter$3) and tos	USPAT	2002/11/06 15:46
-	52	packet adj classifier$1	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:05
-	26	(packet adj classifier$1) and filter$3	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:05
-	21	(packet adj classifier$1) and (qos or tos)	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:05

-	15	((packet adj classifier$1) and filter$3) and ((packet adj classifier$1) and (qos or tos))	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:14
-	4	("5636371" | "5729685" | "5802286" | "5826014").PN.	USPAT	2002/11/14 16:13
-	68	witting.inv.	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:14
-	2	witting.inv. and filter$3	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:15
-	1	witting.inv. and packet$1	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:15
-	253	barzilai	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:15
-	17	barzilai.inv.	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:16
-	77	barzilai and packet$1	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:16
-	9	(barzilai and packet$1) and classif$9	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:18
-	16	"5040176"	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:18
-	2	5040176.pn.	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:20
-	5	tsipora and barzilai.inv.	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:20
-	0	(tsipora and barzilai.inv.) and filter$3	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:21
-	0	(tsipora and barzilai.inv.) and qos	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:21
-	1	(tsipora and barzilai.inv.) and classification	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:24
-	13	wittig and hartmut	USPAT; EPO; DERWENT; USOCR	2002/11/14 16:29
-	3	(("6084879") or ("6335935") or ("6157955")).PN.	USPAT	2002/11/14 16:31
-	522	packet adj classif$8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:12

-	287	(nic or (network adj (interface or adapter))) near5 filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	60477	370/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:09
-	74	370/$.ccls. and ((nic or (network adj (interface or adapter))) near5 filter$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:10
-	8	(370/$.ccls. and ((nic or (network adj (interface or adapter))) near5 filter$3)) and classif$8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:10
-	13	(packet adj classif$8) and ((nic or (network adj (interface or adapter))) near5 filter$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:10
-	1303	(nic or (network adj (interface or adapter or controller))) same filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	1511	(frame or packet or cell) adj classif$8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	311	(nic or (network adj2 (interface or adapter))) near5 filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:13
-	1466	(nic or (network adj2 (interface or adapter or controller))) same filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:14
-	15	((nic or (network adj2 (interface or adapter))) near5 filter$3) and ((frame or packet or cell) adj classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:16
-	7	("5774660" | "5918021" | "6151297" | "6160544" | "6243360" | "6253334" | "6314525").PN.	USPAT	2003/04/30 11:15
-	4	(("5774660" | "5918021" | "6151297" | "6160544" | "6243360" | "6253334" | "6314525").PN.) and filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 11:18
-	25	((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) adj classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:43
-	25	((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) adj classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:44

-	74	((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) same classif$8)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 12:46
-	22	(((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) same classif$8)) and @ad<19981201	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:20
-	18	vaughn.xa. and william and rinehart.xp.	USPAT; EPO; DERWENT; USOCR	2003/04/30 19:15
-	2	6389479.pn.	USPAT; EPO; DERWENT; USOCR	2003/04/30 19:15
-	204094	6389479.pn. and filter$3 or classif$8	USPAT; EPO; DERWENT; USOCR	2003/04/30 19:16
-	1	6389479.pn. and filter$3 and classif$8	USPAT; EPO; DERWENT; USOCR	2003/04/30 19:19
-	1	6078957.pn. and filter$3	USPAT; EPO; DERWENT; USOCR	2003/04/30 19:20
-	22	(((nic or (network adj2 (interface or adapter or controller))) same filter$3) and ((frame or packet or cell) same classif$8)) and @ad<19981201	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:27
-	1466	((nic or (network adj2 (interface or adapter or controller))) same filter$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:27
-	8260	709/22$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:27
-	132	709/22$.ccls. and (((nic or (network adj2 (interface or adapter or controller))) same filter$3))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:27
-	60477	370/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:28
-	944	370/$.ccls. and 709/22$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:28
-	27	(370/$.ccls. and 709/22$.ccls.) and (709/22$.ccls. and (((nic or (network adj2 (interface or adapter or controller))) same filter$3)))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:32
-	36	(packet adj classif$8) and nic	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:33

-	4	((packet adj classif$8) and nic) and @ad<19981101	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 19:34
-	2	(((packet adj classif$8) and nic) and @ad<19981101) and filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 20:41
-	10	(policy adj server) same filter$1	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 20:44
-	4	vaughn.xa. and profile and william	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 20:52
-	1	6510164.pn. and filter$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/04/30 20:52
-	6943	classif$9 same filter$1	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:57
-	41	barzilai.inv.	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:57
-	1	(classif$9 same filter$1) and barzilai.inv.	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:57
-	420945	(classif$9 same filter$1) samd dynamic$4	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:57
-	152	(classif$9 same filter$1) same dynamic$4	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:58
-	432264	filter$3 same (expir$8 or delet$4 or remov$4 or creat$4)	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:58
-	11494	classif$9 same (packet$1 or message$1 or frame$1 or datagram$1)	USPAT; EPO; DERWENT; USOCR	2004/01/16 08:59
-	12709	admission same (polic$3 or regulat$4 or rule$1 or criteria)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:00
-	70379	(traffic or packet$1 or frame$1 or datagram$1) same (filter$1 or shaping)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:06
-	15	((traffic or packet$1 or frame$1 or datagram$1) same (filter$1 or shaping)) and (admission same (polic$3 or regulat$4 or rule$1 or criteria)) and (classif$9 same (packet$1 or message$1 or frame$1 or datagram$1)) and (filter$3 same (expir$8 or delet$4 or remov$4 or creat$4)) and (classif$9 same filter$1)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:02

-	1093	terrell.inv.	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:02
-	2	terrell.inv. and (nortel.asn. or (bay adj network$1.asn.))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:03
-	5748	(nortel.asn. or (bay adj network$1.asn.))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:03
-	10	((nortel.asn. or (bay adj network$1.asn.))) and (classif$9 same filter$1)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:04
-	8	("5313455" | "5463620" | "5781532" | "6104700" | "6167027" | "6188698" | "6222844" | "6381649").PN.	USPAT	2004/01/16 09:05
-	581148	(traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:09
-	7	("5546390" | "5873078" | "6029170" | "6223174" | "6233574" | "6298340" | "6396842").PN.	USPAT	2004/01/16 09:07
-	11	("5243596" | "5406322" | "5509006" | "5721920" | "5862335" | "5890217" | "6104696" | "6154446" | "6157955" | "6167047" | "6240452").PN.	USPAT	2004/01/16 09:07
-	7	6279035.URPN.	USPAT	2004/01/16 09:08
-	7	("5600820" | "5828844" | "5878043" | "5892924" | "5920705" | "5926459" | "5949786").PN.	USPAT	2004/01/16 09:09
-	0	(("5600820" | "5828844" | "5878043" | "5892924" | "5920705" | "5926459" | "5949786").PN.) and (classif$9 same filter$1)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:09
-	3	(("5600820" | "5828844" | "5878043" | "5892924" | "5920705" | "5926459" | "5949786").PN.) and (classif$9 same (packet$1 or message$1 or frame$1 or datagram$1))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:10
-	2	((("5600820" | "5828844" | "5878043" | "5892924" | "5920705" | "5926459" | "5949786").PN.) and (classif$9 same (packet$1 or message$1 or frame$1 or datagram$1))) and filter$3	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:11
-	759	(filter$3 same (expir$8 or delet$4 or remov$4 or creat$4)) and (classif$9 same filter$1) and ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:11
-	30	((filter$3 same (expir$8 or delet$4 or remov$4 or creat$4)) and (classif$9 same filter$1) and ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy))) and (admission same (polic$3 or regulat$4 or rule$1 or criteria))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:16
-	34	6167445.URPN.	USPAT	2004/01/16 09:15
-	13	6031841.URPN.	USPAT	2004/01/16 09:16
-	152	((classif$9 same filter$1) same dynamic$4) and (classif$9 same filter$1)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:16

-	46	((classif$9 same filter$1) same dynamic$4) and ((traffic or packet$1 or frame$1 or datagram$1) same (filter$1 or shaping))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:16
-	22	(((classif$9 same filter$1) same dynamic$4) and ((traffic or packet$1 or frame$1 or datagram$1) same (filter$1 or shaping))) and @ad<19981228	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:18
-	22387	dynamic$4 same filter$3	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:19
-	1138	(dynamic$4 same filter$3) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:19
-	252	((dynamic$4 same filter$3) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy))) and (709/$.ccls. or 370/$.ccls.)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:26
-	150	(((dynamic$4 same filter$3) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy))) and (709/$.ccls. or 370/$.ccls.)) and @ad<19981228	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:20
-	30	((dynamic$4 same filter$3) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy))) and (admission same (polic$3 or regulat$4 or rule$1 or criteria))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:21
-	4447	creat$4 near5 dynamic$4 near5 creat$4	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:21
-	127	(creat$4 near5 dynamic$4 near5 creat$4) same filter$3	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:21
-	80	((creat$4 near5 dynamic$4 near5 creat$4) same filter$3) and @ad<19981228	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:26
-	521	(classif$9 same filter$1) and ((classif$9 same (packet$1 or message$1 or frame$1 or datagram$1)) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy)))	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:26
-	93	((classif$9 same filter$1) and ((classif$9 same (packet$1 or message$1 or frame$1 or datagram$1)) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy)))) and (709/$.ccls. or 370/$.ccls.)	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:28
-	46	(((classif$9 same filter$1) and ((classif$9 same (packet$1 or message$1 or frame$1 or datagram$1)) same ((traffic or packet$1 or frame$1 or datagram$1 or admission) same (filter$1 or shaping or control$4 or policy)))) and (709/$.ccls. or 370/$.ccls.)) and @ad<19981228	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:28
-	591	(classif$9 near10 packet$1).ti. or (classif$9 near10 packet$1).ab.	USPAT; EPO; DERWENT; USOCR	2004/01/16 09:28

US006678248B1

(12) **United States Patent**
Haddock et al.

(10) **Patent No.: US 6,678,248 B1**
(45) **Date of Patent: Jan. 13, 2004**

(54) **POLICY BASED QUALITY OF SERVICE**

(75) Inventors: **Stephen R. Haddock**, Los Gatos, CA (US); **Justin N. Chueh**, Palo Alto, CA (US); **Shehzad T. Merchant**, Mountain View, CA (US); **Andrew H. Smith**, Palo Alto, CA (US); **Michael Yip**, Sunnyvale, CA (US)

(73) Assignee: **Extreme Networks**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 772 days.

(21) Appl. No.: **09/597,878**

(22) Filed: **Jun. 20, 2000**

Related U.S. Application Data

(63) Continuation of application No. 09/018,103, filed on Feb. 3, 1998, now Pat. No. 6,104,700.
(60) Provisional application No. 60/057,371, filed on Aug. 29, 1997.

(51) **Int. Cl.**[7] **H04L 12/56**
(52) **U.S. Cl.** **370/235**; 370/412; 370/429
(58) **Field of Search** 370/235, 412, 370/429

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,381,413 A	1/1995	Tobagi et al.	
5,394,402 A	2/1995	Ross	370/94.1
5,499,238 A	3/1996	Shon	370/60.1
5,570,360 A	10/1996	Klausmeier et al.	370/412
5,629,937 A	5/1997	Hayter et al.	370/412
5,742,772 A	4/1998	Sreenan	395/200.56
5,748,614 A	5/1998	Wallmeier	370/412
5,796,719 A	8/1998	Peris et al.	370/418
5,850,399 A	12/1998	Ganmukhi et al.	370/412
5,859,835 A	1/1999	Varma et al.	370/412

OTHER PUBLICATIONS

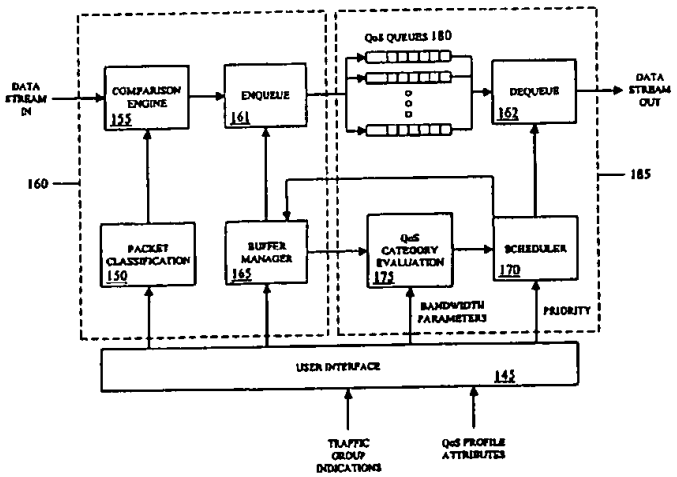
International Search Report, PCT/US98/17817, 4 pages.

Primary Examiner—Melvin Marcelo
(74) *Attorney, Agent, or Firm*—Blackely Sokoloff Taylor & Zafman LLP

(57) **ABSTRACT**

A flexible, policy-based, mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true quality of service (QoS) is provided. According to one aspect of the present invention, a method is provided for managing bandwidth allocation in a network that employs a non-deterministic access protocol, such as an Ethernet network. A packet forwarding device receives information indicative of a set of traffic groups, such as: a MAC address, or IEEE 802.1p priority indicator or 802.1Q frame tag, if the QoS policy is based upon individual station applications; or a physical port if the QoS policy is based purely upon topology. The packet forwarding device additionally receives bandwidth parameters corresponding to the traffic groups. After receiving a packet associated with one of the traffic groups on a first port, the packet forwarding device schedules the packet for transmission from a second port based upon bandwidth parameters corresponding to the traffic group with which the packet is associated. According to another aspect of the present invention, a method is provided for managing bandwidth allocation in a packet forwarding device. The packet forwarding device receives information indicative of a set of traffic groups. The packet forwarding device additionally receives information defining a QoS policy for the traffic groups. After a packet is received by the packet forwarding device, a traffic group with which the packet is associated is identified. Subsequently, rather than relying on an end-to-end signaling protocol for scheduling, the packet is scheduled for transmission based upon the QoS policy for the identified traffic group.

28 Claims, 5 Drawing Sheets



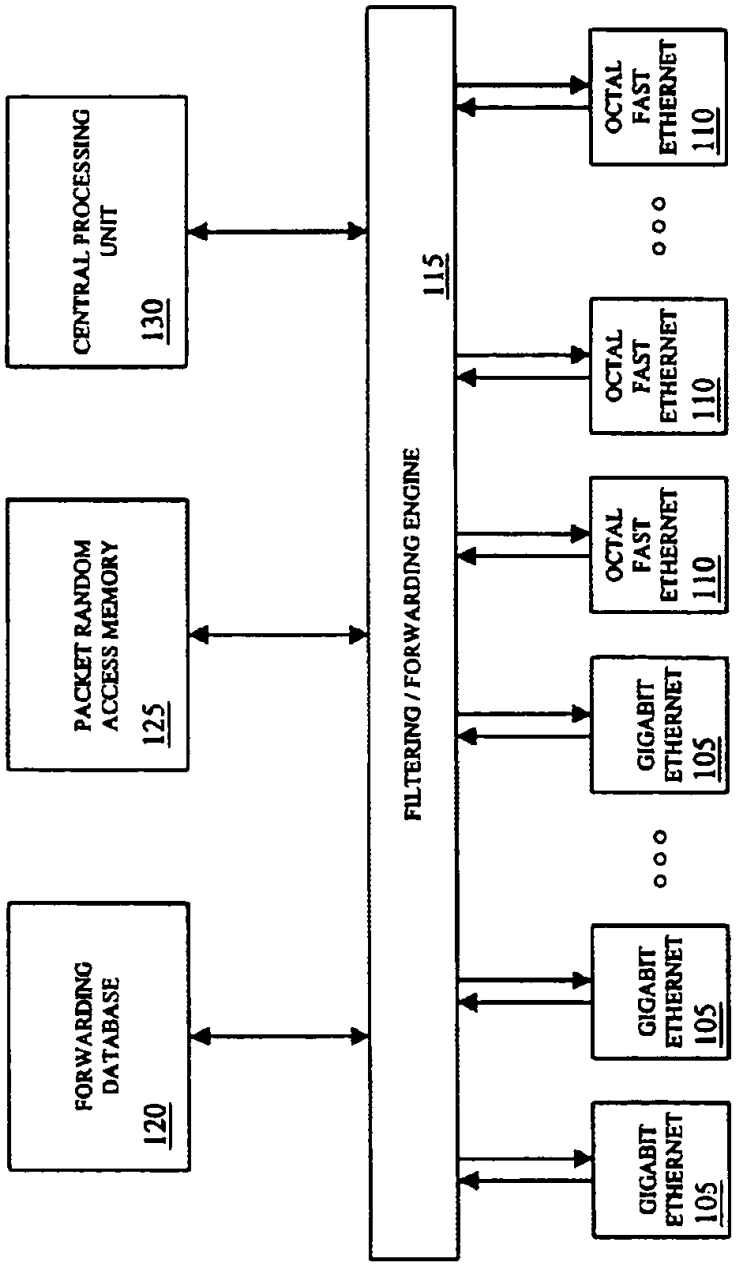

CENTRAL PROCESSING UNIT
130
PACKET RANDOM ACCESS MEMORY
125
FORWARDING DATABASE
120
FILTERING / FORWARDING ENGINE
115
OCTAL FAST ETHERNET
110
OCTAL FAST ETHERNET
110
OCTAL FAST ETHERNET
110
GIGABIT ETHERNET
105
GIGABIT ETHERNET
105
GIGABIT ETHERNET
105
100

FIG. 1A

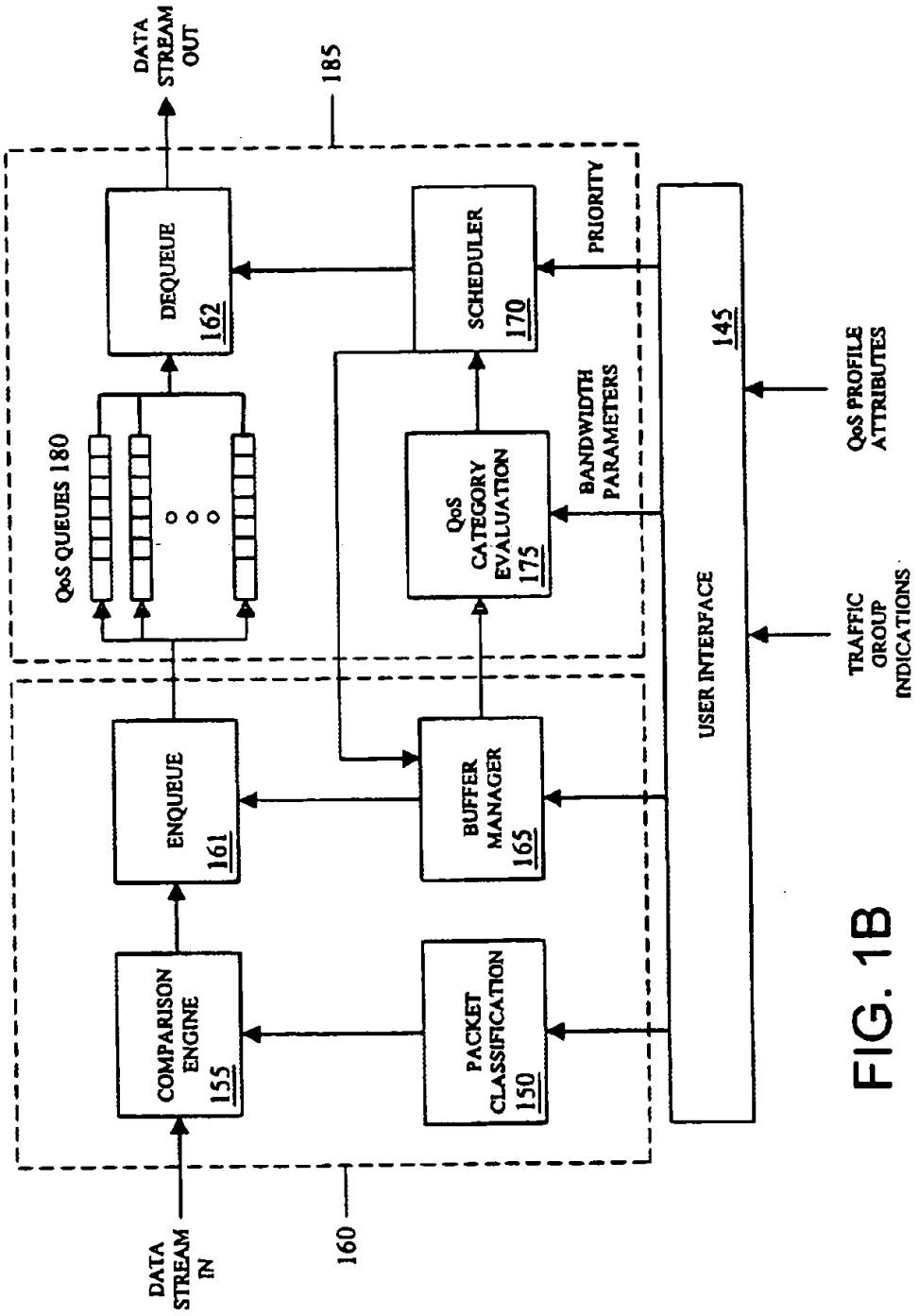

DATA STREAM OUT
185
DEQUEUE 162
SCHEDULER 170
PRIORITY
QoS QUEUES 180
QoS CATEGORY EVALUATION 175
BANDWIDTH PARAMETERS
145
QoS PROFILE ATTRIBUTES
ENQUEUE 161
BUFFER MANAGER 165
USER INTERFACE
TRAFFIC GROUP INDICATIONS
COMPARISON ENGINE 155
PACKET CLASSIFICATION 150
DATA STREAM IN
160

FIG. 1B

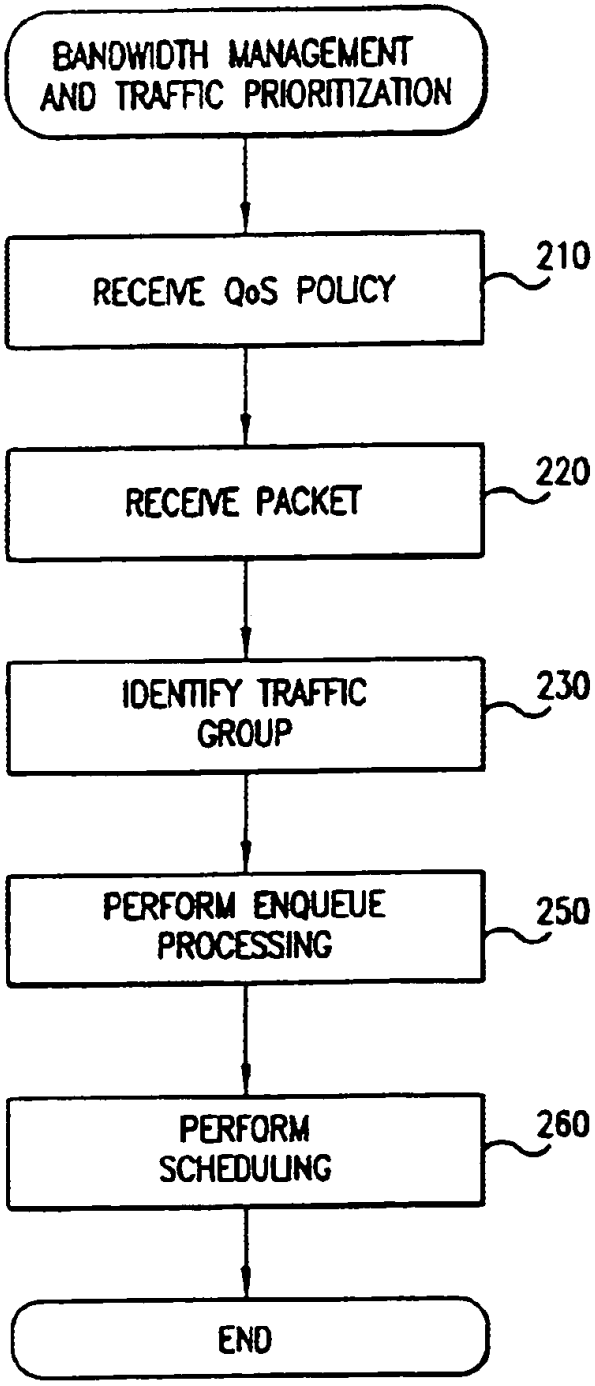

BANDWIDTH MANAGEMENT AND TRAFFIC PRIORITIZATION
RECEIVE QoS POLICY
210
RECEIVE PACKET
220
IDENTIFY TRAFFIC GROUP
230
PERFORM ENQUEUE PROCESSING
250
PERFORM SCHEDULING
260
END

FIG.2

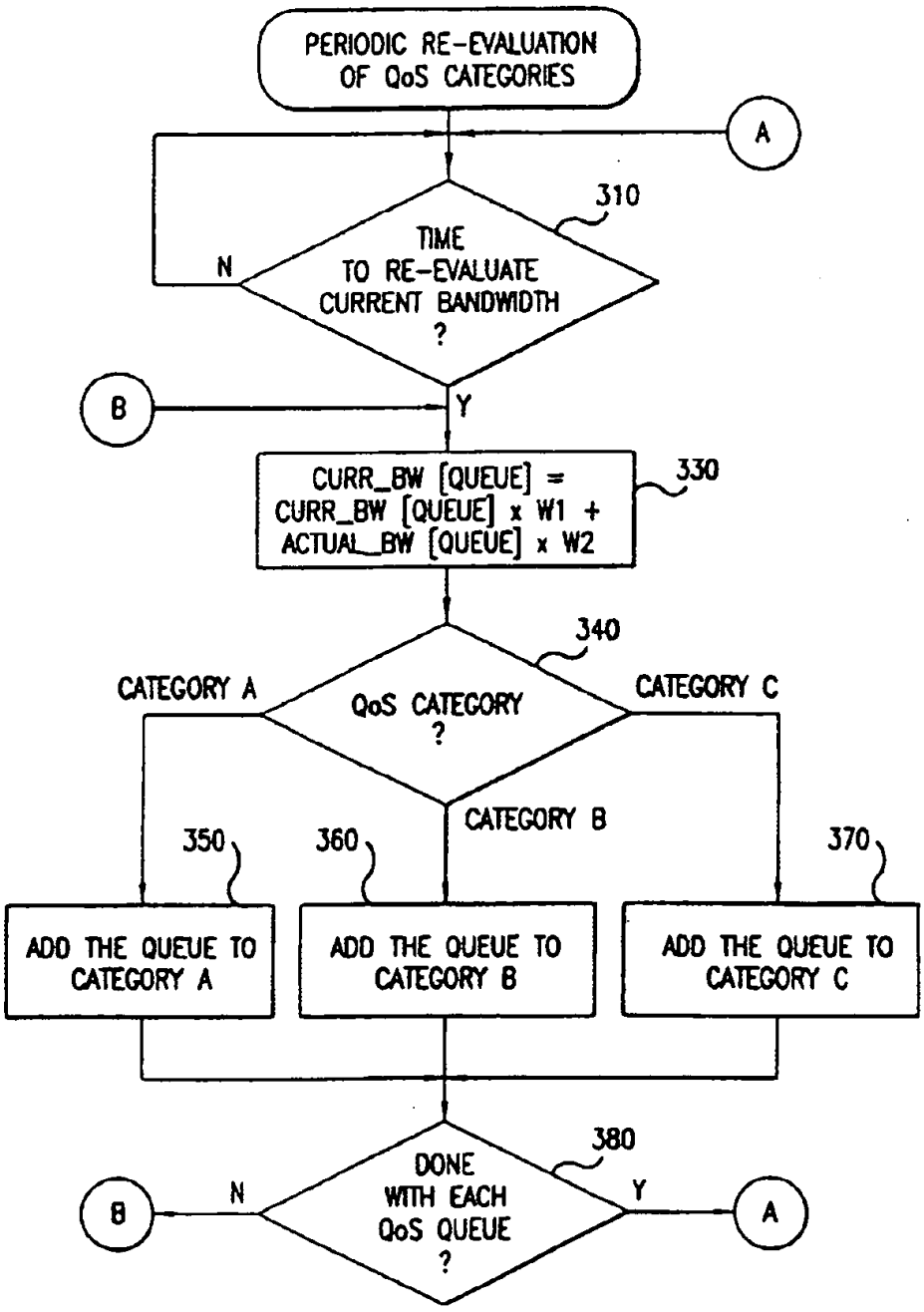

PERIODIC RE-EVALUATION OF QoS CATEGORIES
A
310
TIME TO RE-EVALUATE CURRENT BANDWIDTH ?
N
Y
B
CURR_BW [QUEUE] = CURR_BW [QUEUE] x W1 + ACTUAL_BW [QUEUE] x W2
330
340
QoS CATEGORY ?
CATEGORY A
CATEGORY B
CATEGORY C
350
360
370
ADD THE QUEUE TO CATEGORY A
ADD THE QUEUE TO CATEGORY B
ADD THE QUEUE TO CATEGORY C
380
DONE WITH EACH QoS QUEUE ?
N
Y
B
A

FIG.3

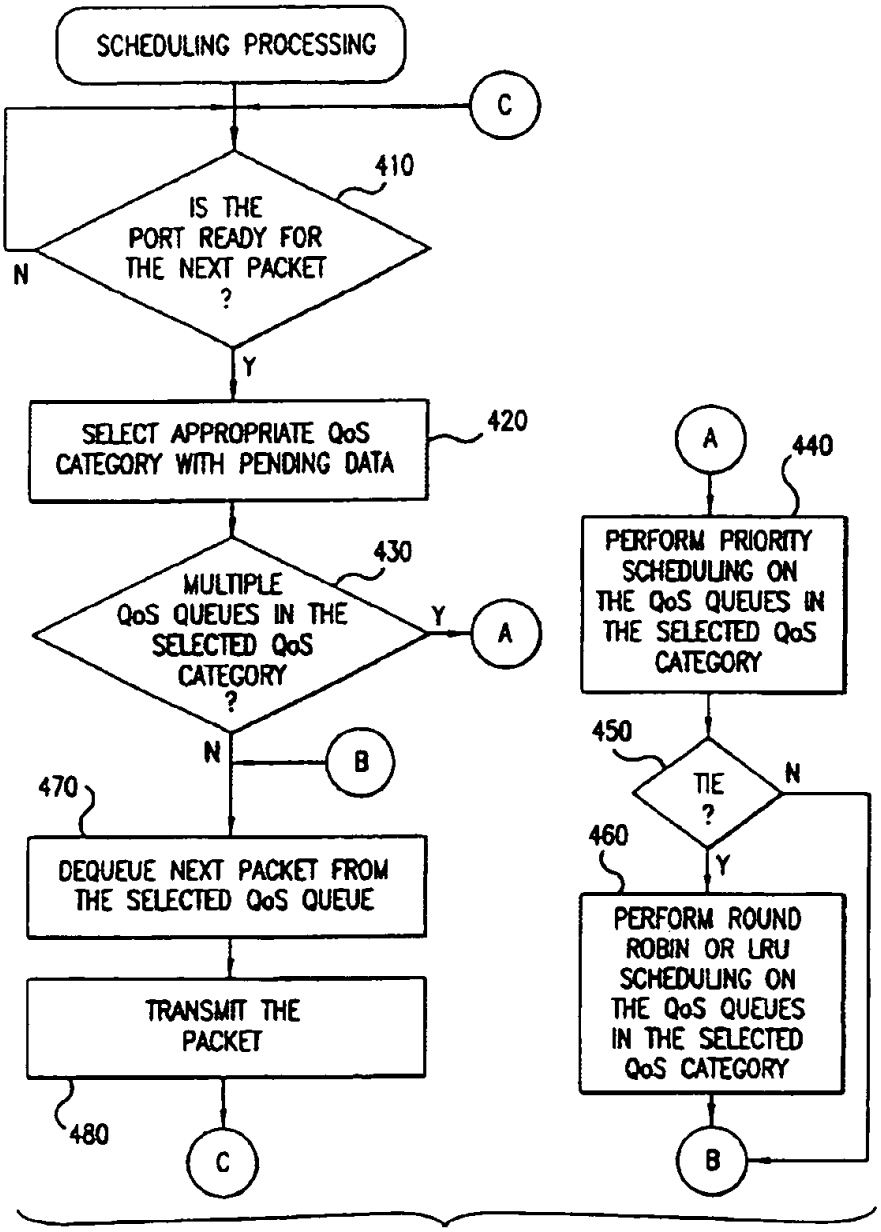

SCHEDULING PROCESSING
C
410
IS THE PORT READY FOR THE NEXT PACKET ?
N
Y
SELECT APPROPRIATE QoS CATEGORY WITH PENDING DATA
420
430
MULTIPLE QoS QUEUES IN THE SELECTED QoS CATEGORY ?
Y
A
N
B
470
DEQUEUE NEXT PACKET FROM THE SELECTED QoS QUEUE
TRANSMIT THE PACKET
480
C
A
440
PERFORM PRIORITY SCHEDULING ON THE QoS QUEUES IN THE SELECTED QoS CATEGORY
450
TIE ?
N
460
Y
PERFORM ROUND ROBIN OR LRU SCHEDULING ON THE QoS QUEUES IN THE SELECTED QoS CATEGORY
B

FIG.4

POLICY BASED QUALITY OF SERVICE

This is a continuation of application Ser. No. 09/018,103, filed on Feb. 3, 1998, now issued U.S. Pat. No. 6,104,700.

This application claims the benefit of U.S. Provisional Application No. 60/057,371, filed Aug. 29, 1997.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of computer networking devices. More particularly, the invention relates to a flexible, policy-based mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true Quality of Service (QoS).

2. Description of the Related Art

Network traffic today is more diverse and bandwidth-intensive than ever before. Today's intranets are expected to support interactive multimedia, full-motion video, rich graphic images and digital photography. Expectations about the quality and timely presentation of information received from networks is higher than ever. Increased network speed and bandwidth alone will not satisfy the high demands of today's intranets.

The Internet Engineering Task Force (IETF) is working on a draft standard for the Resource Reservation Protocol (RSVP), an Internet Protocol-(IP) based protocol that allows end-stations, such as desktop computers, to request and reserve resources within and across networks. Essentially, RSVP is an end-to-end protocol that defines a means of communicating the desired Quality of Service between routers. RSVP is receiver initiated. The end-station that is receiving the data stream communicates its requirements to an adjacent router and those requirements are passed back to all intervening routers between the receiving end-station and the source of the data stream and finally to the source of the data stream itself. Therefore, it should be apparent that RSVP must be implemented across the whole network. That is, both end-stations (e.g., the source and destination of the data stream) and every router in between should be RSVP compliant in order to accommodate the receiving end-station's request.

While RSVP allows applications to obtain some degree of guaranteed performance, it is a first-come, first-served protocol, which means if there are no other controls within the network, an application using RSVP may reserve and consume resources that could be needed or more effectively utilized by some other mission-critical application. A further limitation of this approach to resource allocation is the fact that end-stations and routers must be altered to be RSVP compliant. Finally, RSVP lacks adequate policy mechanisms for allowing differentiation between various traffic flows. It should be appreciated that without a policy system in place, the network manager loses control.

Recent attempts to facilitate traffic differentiation and prioritization include draft standards specified by the Institute of Electrical and Electronics Engineers (IEEE). The

IEEE 802.1Q draft standard provides a packet format for an application to specify which Virtual Local Area Network (VLAN) a packet belongs to and the priority of the packet. The IEEE 802.1p committee provides a guideline to classify traffic based on a priority indicator in an 802.1Q frame tag. This allows VLANs to be grouped into eight different traffic classes or priorities. The IEEE 802.1p committee does not, however, define the mechanism to service these traffic classes.

What is needed is a way to provide true Quality of Service ("QoS") in a network employing a non-deterministic access protocol, such as an Ethernet network, that not only has the ability to prioritize and service different traffic classes, but additionally provides bandwidth management and guarantees a quantifiable measure of service for packets associated with a particular traffic class. More specifically, with respect to bandwidth management, it is desirable to employ a weighted fair queuing delivery schedule which shares available bandwidth so that high priority traffic is usually sent first, but low priority traffic is still guaranteed an acceptable minimum bandwidth allocation. Also, it is desirable to centralize the control over bandwidth allocation and traffic priority to allow for QoS without having to upgrade or alter end-stations and existing routers as is typically required by end-to-end protocol solutions. Further, it would be advantageous to put the control in the hands of network managers by performing bandwidth allocation and traffic prioritization based upon a set of manager-defined administrative policies. Finally, since there are many levels of control a network manager may elect to administer, it is desirable to provide a variety of scheduling mechanisms based upon a core set of QoS profile attributes.

BRIEF SUMMARY OF THE INVENTION

A flexible, policy-based, mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true Quality of Service (QoS) is described. According to one aspect of the present invention, a method is provided for managing bandwidth allocation in a network that employs a non-determninistic access protocol. A packet forwarding device receives information indicative of a set of traffic groups. The packet forwarding device additionally receives parameters, such as bandwidth and priority parameters, corresponding to the traffic groups. After receiving a packet associated with one of the traffic groups on a first port, the packet forwarding device schedules the packet for transmission from a second port based upon parameters corresponding to the traffic group with which the packet is associated. Advantageously, in this manner, a weighted fair queuing schedule that shares bandwidth according to some set of rules may be achieved.

According to another aspect of the present invention, a method is provided for managing bandwidth allocation and traffic prioritization in a packet forwarding device. The packet forwarding device receives information indicative of a set of traic groups. The packet forwarding device additionally receives information defining a Quality of Service (QoS) policy for the traffic groups. After a packet is received by the packet forwarding device, a traffic group with which the packet is associated is identified. Subsequently, rather than relying on an end-to-end signaling protocol for scheduling, the packet is scheduled for transmission based upon the QoS policy for the identified traffic group. Therefore, bandwidth allocation and traffic prioritization are based upon a set of administrative policies over which the network manager retains control.

According to yet another aspect of the present invention, a number of QoS queues are provided at each port of the

packet forwarding device. A current bandwidth metric is determined for each of the QoS queues for a particular port. The QoS queues are divided into two groups based upon their respective bandwidth metrics and their respective minimum bandwidth requirements. Subsequently, the groups are used as a first level arbitration mechanism to select a QoS queue that will source the next packet.

Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. 1A is a simplified block diagram of an exemplary switch architecture in which one embodiment of the present invention may be implemented.

FIG. 1B is a logical view of the interaction between switch processing blocks according to one embodiment of the present invention.

FIG. 2 is a flow diagram illustrating high level bandwidth management and traffic prioritization processing according to one embodiment of the present invention.

FIG. 3 is a flow diagram illustrating periodic evaluation of QoS categories according to one embodiment of the present invention.

FIG. 4 is a flow diagram illustrating next packet scheduling according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A flexible, policy-based, mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true Quality of Service (QoS) is described. "Quality of Service" in this context essentially means that there is a quantifiable measure of the service being provided. The measure of service being provided may be in terms of a packet loss rate, a maximum delay, a committed minimum bandwidth, or a limited maximum bandwidth, for example.

In the present invention, a number of QoS queues may be provided at each port of a packet forwarding device, such as a Local Area Network (LAN) switch. Based upon a set of QoS parameters, various types of traffic can be distinguished and associated with particular QoS queues. For example, packets associated with a first traffic group may be placed onto a first QoS queue and packets associated with another traffic group may be placed onto a second QoS queue. When a port is ready to transmit the next packet, a scheduling mechanism may be employed to select which QoS queue of the QoS queues associated with the port will provide the next packet for transmission.

In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

The present invention includes various steps, which will be described below. The steps of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor programmed with the instructions to perform the steps. Alternatively, the steps may be performed by a combination of hardware and software. While, embodiments of the present invention will be described with reference to a high speed Ethernet switch, the method and apparatus described herein are equally applicable to other types of network devices or packet forwarding devices.

An Exemplary Switch Architecture

An overview of the architecture of a switch **100** in which one embodiment of the present invention may be implemented is illustrated by FIG. 1A. The central memory architecture depicted includes multiple ports **105** and **110** each coupled via a channel to a filtering/forwarding engine **115**. Also coupled to the filtering/forwarding engine **115** is a forwarding database **120**, a packet Random Access Memory (RAM) **125**, and a Central Processing Unit (CPU) **130**.

According to one embodiment, each channel is capable of supporting a data transfer rate of one gigabit per second in the transmit direction and one gigabit per second in the receive direction, thereby providing 2 Gb/s full-duplex capability per channel. Additionally, the channels may be configured to support one Gigabit Ethernet network connection or eight Fast Ethernet network connections.

The filtering/forwarding engine **115** includes an address filter (not shown), a switch matrix (not shown), and a buffer manager (not shown). The address filter may provide bridging, routing, Virtual Local Area Network (VLAN) tagging functions, and traffic classification. The switch matrix connects each channel to a central memory such as packet RAM **125**. The buffer manager controls data buffers and packet queue structures and controls and coordinates accesses to and from the packet RAM **125**.

The forwarding database **120** may store information useful for making forwarding decisions, such as layer **2** (e.g., Media Access Control (MAC) layer), layer **3** (e.g., Network layer), and/or layer **4** (e.g., Transport layer) forwarding information, among other things. The switch **100** forwards a packet received at an input port to an output port by performing a search on the forwarding database using address information contained within the header of the received packet. If a matching entry is found, a forwarding decision is constructed that indicates to which output port the received packet should be forwarded, if any. Otherwise, the packet is forwarded to the CPU **130** for assistance in constructing a forwarding decision.

The packet RAM **125** provides buffering for packets and acts as an elasticity buffer for adapting between incoming and outgoing bandwidth differences. Packet buffering is discussed further below.

Logical View of Exemplary Switch Processing

FIG. 1B is a logical view of the interaction between exemplary switch processing blocks that may be distributed throughout the switch **100**. For example, some of the processing may be performed by functional units within the ports of the switch and other processing may be performed by the CPU **130** or by the address filter/switch matrix/buffer manager **115**. In any event, the processing can be conceptually divided into a first group of functions **160** dedicated to input processing and a second group of functions **185** dedicated to output processing. According to the present embodiment, the first group **160** includes a comparison

engine **155**, an enqueue block **161**, a packet classification block **150**, and a buffer manager **165**. The second group **185** includes a dequeue block **162**, a Quality of Service (QoS) category evaluation block **175**, and a scheduler **170**.

Additionally, a user interface (UI) **145** may be provided for receiving various parameters from the network manager. The UI may be text based or graphical. In one embodiment, the UI **145** may include an in-band HyperText Markup Language (HTML) browser-based management tool which may be accessed by any standard web browser. In any event, the goal of the UI **145** is to separate high-level policy components, such as traffic grouping and QoS profiles from the details of the internal switch hardware. Thus, user configuration time is minimized and a consistent interface is provided to the user.

The UI **145** receives information indicative of one or more traffic groups. This information may be provided by the network manager. There are several ways to define a traffic group. Table 1 below illustrates a variety of traffic classification schemes that may be supported by the UI **145**.

TABLE 1

	Traffic Classification	
Policy Based Upon	Traffic Group Definition	OSI Layer
Applications	TCP Session UDP Session RSVP Flow	Transport Layer
Network Layer	Network Layer Protocol	Network Layer
Topology or Groups of Users	Subnet or IP Address VLAN Identifier	
End-Station Applications	MAC Address 802.1p or 802.1Q	Link Layer
Physical Topology	Physical Port	Physical Layer

The information used to identify a traffic group typically depends upon what terms the QoS policy is defined. If the QoS policy is based on applications, traffic groups may be differentiated at the Transport layer by Transmission Control Protocol (TCP) session or User Datagram Protocol (UDP) session. For example, the network manager may provide information indicative of TCP source and destination ports and IP source and destination addresses to identify traffic groups. However, if the QoS policy is based upon the Network layer topology or groups of users, traffic group definition may be more convenient by supplying information regarding the Network layer protocol, such as Internet Protocol (IP) or Internetwork Packet Exchange (IPX), the subnet or IP addresses, or VLAN identifiers. If the QoS policy is defined by end-station applications, then Media Access Control (MAC) addresses, IEEE 802.1p priority indications, or IEEE 802.1Q frames may be employed to identify traffic groups. Finally, if the QoS policy is physical topology based, physical port identifiers may be used to differentiate traffic groups.

It should be noted that Table 1 merely presents an exemplary set of traffic group identification mechanisms. From the examples presented herein, additional, alternative, and equivalent traffic grouping schemes and policy considerations will be apparent to those of ordinary skill in the art. For example, other state information may be useful for purposes of packet classification, such as the history of previous packets, the previous traffic load, the time of day, etc.

It is appreciated that traffic classifications based upon the traffic group definitions listed above may result in overlap. Should the network manager define overlapping traffic groups, the UI **145** may issue an error message and reject the most recent traffic group definition, the UI **145** may issue a warning message to the network manager and allow the more specific traffic group definition to override a conflicting general traffic group definition, or the UI **145** may be configured to respond in another manner.

A number of QoS queues **180** may be provided at each of the ports of a packet forwarding device. In one embodiment, a mapping of traffic groups to QoS queues **180** may be maintained. As traffic groups are provided by the network manager, the UI **145** updates the local mapping of traffic groups to QoS queues **180**. This mapping process may be a one-to-one mapping of the traffic groups defined by the network manager to the QoS queues **180** or the mapping process may be more involved. For example, there may be more traffic groups than QoS queues **180**, in which case, more than one traffic group will be mapped to a single QoS queue. Some consolidation rules for combining multiple traffic groups into a single QoS queue will be discussed below.

At any rate, by providing a layer of abstraction in this manner, the network manager need not be burdened with the underlying implementation details, such as the number of QoS queues per port and other queuing parameters. Another advantage achieved by this layer of abstraction between the traffic group definitions and the physical QoS queues is the fact that the UI **145** is now decoupled from the underlying implementation. Therefore, the UI **145** need not be updated if the hardware QoS implementation changes. For example, software providing for traffic group definition need not be changed simply because the number of QoS queues per port provided by the hardware changes.

The input data stream is received by the comparison engine **155** from input switch ports (not shown). Under the direction of the packet classification process **150**, the comparison engine **155** determines with which of the previously defined traffic groups a packet in the data stream is associated. The packet classification block **150** may employ the traffic group indications provided by the network manager to provide the comparison engine **155** with information regarding locations and fields to be compared or ignored within the header of a received packet, for example. It should be appreciated if the comparison required for traffic classification is straightforward, such as in a conventional packet forwarding device, then the comparison engine **155** and the packet classification block **150** may be combined.

The packet classification block **150** in conjunction with the UI **145** provide a network manager with a flexible mechanism to control traffic prioritization and bandwidth allocation through the switch **100**. Importantly, no end-to-end signaling protocol needs to be implemented by the network devices. For example, the end-station that is to receive the data stream need not reserve bandwidth on each of the intermediate devices between it and the source of the data stream. Rather, a packet forwarding device employing the present invention can provide some benefit to the network without requiring routers and/or end-stations to do anything in particular to identify traffic. Thus, traffic priority may be enforced by the switch **100** and QoS may be delivered to applications without altering routers or end-stations.

According to one embodiment, the buffer manager **165** participates in policy based QoS by controlling the allocation of buffers within the packet RAM **125**. Buffers may be dynamically allocated to QoS queues **180** as needed, within constraints established by QoS profile attributes, which are

discussed below. The buffer manager **165** may maintain several programmable variables for each QoS queue. For example, a Minimum Buffer Allocation and a Maximum Queue Depth may be provided for each QoS queue. The Minimum Buffer Allocation essentially reserves some minimum number of buffers in the packet RAM **125** for the QoS queue with which it is associated. The Maximum Queue Depth establishes the maximum number of buffers that can be placed on a given QoS queue. The buffer manager **165** also maintains a Current Queue Depth for each QoS queue to assure the maximum depth is not exceeded. For example, before allowing a buffer to be added to a given QoS queue, the buffer manager **165** may compare the Maximum Queue Depth to the Current Queue Depth to ensure the Maximum Queue Depth is not exceeded.

Variables are also maintained for tracking free buffers in the packet RAM **125**. At initialization, a Buffers Free Count contains the total number of buffers available in the packet RAM **125** and a Buffers Reserved Count contains the sum of the minimum buffer allocations for the QoS queues **180**. As packets are received they are stored in free buffers, and the Buffers Free Count is decremented by the number of buffers used for such storage. After the appropriate QoS queue has been identified the buffer manager **165** instructs the enqueue block **161** to add the packet to the QoS queue. The enqueue block **161** links the packet to the identified queue provided that the Current Queue Depth is less than the Maximum Queue Depth and either (1) the Current Queue Depth is less than the Minimum Buffer Allocation or (2) the Buffers Reserved Count is less than the Buffers Free Count. Therefore, if a QoS queue exceeds its reserve of buffers (e.g., Minimum Buffer Allocation), to the extent that additional buffers remain free, the QoS queue may continue to grow. Otherwise, the enqueue block **161** will discard the packet, the buffers are returned to the free pool, and the Buffers Free Count is increased by the number of buffers that would have been consumed by the packet. When a packet is successfully linked to a QoS queue, the Current Queue Depth for that QoS queue is increased by the number of buffers used by the packet. If, prior to the addition of the packet to the queue, the Current Queue Depth was less than the Minimum Buffer Allocation then the Buffers Reserved Count is decreased by the lesser of (1) the number of buffers in the packet or (2) the difference between the Current Queue Depth and the Minimum Buffer Allocation.

The QoS category evaluation process **175** separates the QoS queues into a plurality of categories based upon a set of bandwidth parameters. The scheduler **170** uses the grouping provided by the QoS category evaluation process **175** to select an appropriate QoS queue for sourcing the next packet for a particular port. The evaluation of QoS queue categories may be performed periodically or upon command by the scheduler **170**, for example. Periodic evaluation of QoS categories and scheduling is discussed in further detail below.

Responsive to the scheduler **170** the dequeue block **162** retrieves a packet from a specified QoS queue. After the packet has been transmitted, the buffer variables are updated. The Buffers Free Count is increased and the Current Queue Depth is decreased by the number of buffers utilized to store the packet. If the resulting Current Queue Depth is less than the Minimum Buffer Allocation, then the Buffers Reserved Count is increased by the lesser of the number of buffers utilized to store the packet or the difference between the Current Queue Depth and the Minimum Buffer Allocation.

QoS Profile Attributes

Setting QoS policy is a combination of identifying traffic groups and defining QoS profiles for those traffic groups. According to one embodiment, each individual traffic group may be associated with a QoS profile. However, in alternative embodiments, multiple traffic groups may share a common QoS profile. Having described traffic group classification and identification above, QoS profile attributes (also referred to as parameters) will now be discussed.

Several queuing mechanisms may be implemented using one or more of the following parameters associated with a traffic group: (1) minimum bandwidth, (2) maximum bandwidth, (3) peak bandwidth, (4) maximum delay, and (5) relative priority. In general, the minimum, maximum, and peak bandwidth parameter may be expressed in Mbps, a percentage of total bandwidth, or any other convenient representation.

Minimum bandwidth indicates the minimum amount of bandwidth a particular traffic group needs to be provided over a defined time period. If the sum of the minimum bandwidths for all traffic groups defined is less than 100% of the available bandwidth, then the scheduling processing, discussed below, can assure that each traffic group will receive at least the minimum bandwidth requested.

Maximum bandwidth is the maximum sustained bandwidth the traffic group can realize over a defined time period. In contrast, peak bandwidth represents the bandwidth a traffic group may utilize during a particular time interval in excess of the maximum bandwidth. The peak bandwidth parameter may be used to limit traffic bursts for the traffic group with which it is associated. The peak bandwidth also determines how quickly the traffic group's current bandwidth will converge to the maximum bandwidth. By providing a peak bandwidth value that is much higher than the maximum bandwidth, if sufficient bandwidth is available, the maximum bandwidth will be achieved relatively quickly. In contrast, a peak bandwidth that is only slightly higher that the maximum bandwidth will cause the convergence to the maximum bandwidth to be more gradual.

Maximum delay specifies a time period beyond which further delay cannot be tolerated for the particular traffic group. Packets comprising the traffic group that are forwarded by the switch **100** are guaranteed not to be delayed by more than the maximum delay specified.

Relative priority defines the relative importance of a particular traffic group with respect to other traffic groups. As will be discussed further below, within the same QoS category, traffic groups with a higher priority are preferred over those with lower priorities.

This small set of parameters in combination with the variety of traffic classification schemes gives a network manager enormous control and flexibility in prioritizing and managing traffic flowing through packet forwarding devices in a network. For example, the QoS profile of a video traffic group, identified by UDP session, might be defined to have a high priority and a minimum bandwidth of 5 Mbps, while the QoS profile of an engineering traffic group, identified by VLAN, may be set to a second priority, a minimum bandwidth of 30 Mbps, a maximum bandwidth of 50 Mbps, and a peak bandwidth of 60 Mbps. Concurrently, the QoS profile of a World Wide Web (WWW) traffic group, identified by protocol (e.g., IP), may be set to have a low priority, a minimum bandwidth of 0 Mbps, a maximum bandwidth of 100%, and a peak bandwidth of 100%.

Consolidation Rules

It was mentioned earlier that multiple traffic groups may be mapped to a single QoS queue. This may be accomplished by maintaining an independent set of variables (e.g.,

minimum bandwidth, maximum bandwidth, peak bandwidth, maximum delay, and relative priority) for each QoS queue in addition to those already associated with each traffic group and following the general consolidation rules outlined below.

Briefly, when the mapping from traffic groups to QoS queues is one-to-one, the determination of a particular QoS queues' attributes is straightforward. The QoS queue's attributes simply equal the traffic group's attributes. However, when combining multiple traffic groups that do not share a common QoS profile onto a single QoS queue, the following general consolidation rules are suggested: (1) add minimum attributes of the traffic groups being combined to arrive at an appropriate minimum attribute for the target QoS queue (e.g., the QoS queue in which the traffic will be merged), (2) use the largest of maximum attributes to arrive at an appropriate value for a maximum attribute for the target QoS queue, and (3) avoid merging traffic groups that have different relative priorities. This last rule suggests the number of priority levels provided should be less than or equal to the number of QoS queues supported by the implementation to assure traffic groups with different priorities are not combined in the same QoS queue.

Importantly, when a network manager has determined that multiple traffic groups will share a common QoS profile, the consolidation rules need not apply, as the network manager has already, in effect, manually consolidated the parameters.

Bandwidth Management and Traffic Prioritization

Having described an exemplary environment in which one embodiment of the present invention may be implemented, bandwidth management and traffic prioritization will now be described with reference to FIG. **2**. FIG. **2** is a flow diagram illustrating the high level bandwidth management and traffic prioritization processing according to one embodiment of the present invention. In this embodiment, at step **210**, a manager-defined QoS policy may be received via the UI **145**, for example. The QoS policy is a combination of traffic groups and QoS profile attributes corresponding to those traffic groups.

At step **220**, a packet is received by the switch **100**. Before the packet can be placed onto a QoS queue for transmission, the traffic group to which the packet belongs is identified at step **230**. Typically, information in the packet header, for example, can be compared to the traffic group criteria established by the network manager to identify the traffic group to which the packet belongs. This comparison or matching process may be achieved by programming filters in the switch **100** that allow classification of traffic. According to one embodiment, the packet may be identified using the traffic group definitions listed in Table 1.

At step **250**, enqueue processing is performed. The packet is added to the rear of the appropriate QoS queue for the identified traffic group. Importantly, if a maximum delay has been assigned to the traffic group with which the packet is associated, then the packet should either be dropped or transmitted within the period specified. According to one embodiment, this may be accomplished by limiting the depth (also referred to as length) of the corresponding QoS queue. Given the minimum bandwidth of the QoS queue and the maximum delay the traffic group can withstand, a maximum depth for the QoS queue can be calculated. If the QoS queue length remains less than or equal to the maximum length, then the packet is added to the QoS queue. However, if the QoS queue length would exceed the maximum length by the addition, then the packet is dropped.

At step **260**, scheduling is performed. The scheduling/dequeuing processing involves determining the appropriate QoS queue group, selecting the appropriate QoS queue within that QoS queue group, and removing the packet at the front of the selected QoS queue. This selected packet will be the next packet the port transmits. Scheduling will be discussed further below.

Evaluation of QoS Categories

According to one embodiment of the present invention, it is advantageous to divide the QoS queues into at least two categories. The categories may be defined based upon the maximum bandwidth, the minimum bandwidth, the peak bandwidth, and the "current bandwidth." The current bandwidth should not be mistaken for a bandwidth at an instant in time, rather the current bandwidth is a moving average that is updated periodically upon the expiration of a predetermined time period. Empirical data suggests this predetermined time period should be on the order of ten packet times, wherein a packet time is the time required to transmit a packet. However, depending upon the environment and the nature of the traffic, a value in the range of one to one hundred packet times may be more suitable.

The members of the first category ("Category A") are those QoS queues which have a current bandwidth that is below their peak bandwidth and below their minimum bandwidth. Members of the second category ("Category B") include those QoS queues that have a current bandwidth that is greater than or equal to their minimum bandwidth, but less than both their maximum bandwidth and their peak bandwidth. The remaining QoS queues (e.g., those having a current bandwidth that is greater than or equal to either the peak bandwidth or the maximum bandwidth) are ineligible for transmission. These QoS queues that are ineligible for transmission can be considered a third category ("Category C"). With this overview of QoS categories, an exemplary process for periodic evaluation of QoS categories will now be described.

FIG. **3** is a flow diagram illustrating periodic evaluation of QoS categories according to one embodiment of the present invention. In this embodiment, at step **310**, processing loops until the predetermined evaluation time period has expired. For example, a test may be performed to determine if the current time is greater than or equal to the last evaluation time plus the predetermined evaluation time interval. Alternatively, the evaluation process may be triggered by an interrupt. In any event, when it is time to evaluate the QoS queue categorization, processing continues with step **330**.

It will be appreciated that the time interval chosen for the predetermined evaluation time period should not be too long or too short. If the time interval is too long, one QoS queue might be allowed to monopolize the link until its maximum bandwidth is achieved while other QoS queues remain idle. If the time interval is too short, transmitting a single packet or remaining idle for a single packet time may cause the QoS queue to become a member of a different QoS category (e.g., the single transmission may cause the current bandwidth to exceed the maximum bandwidth or the single idle time may cause the current bandwidth to fall below the minimum bandwidth) because the moving average moves very quickly over short time intervals.

At step **330**, the current bandwidth for a particular QoS queue is set to the current bandwidth for that QoS queue as calculated in the previous time interval multiplied by a first weighting factor plus the actual bandwidth that particular QoS queue received in the previous time interval multiplied

by a second weighting factor, wherein the weighting factors may be selected to achieve the desired level of responsiveness in the current bandwidth metric. For example, it may be desirable to have the current bandwidth converge to within a certain percentage of a sustained bandwidth if that bandwidth has been sustained for a certain amount of time. Exemplary weighting factors are in the form (w−1)/w and 1/w, respectively. Using weighting factors of 15/16 for the first weighting factor and a value of 1/16 for the second weighting factor, for example, the current bandwidth will reflect 50% of a step within 13 time intervals, 80% of a step within 27 time intervals, and will be within 2% of the sustained bandwidth in approximately 63 time intervals (assuming a maximum and peak bandwidth of 100%). Alternative ratios and current bandwidth metrics will be apparent to those of ordinary skill in the art.

After the current bandwidth has been evaluated for a QoS queue, at step **340**, the QoS queue bandwidth parameters can be compared to the current bandwidth to determine to which QoS category the QoS queue belongs. As described above, if (CURR_BW<PEAK_BW) and (CURR_BW<MIN_BW), then the QoS queue is associated with Category A at step **350**. If (CURR_BW≧MIN_BW) and ((CURR_BW<MAX_BW) and (CURR_BW<PEAK_BW)), then the QoS queue is associated with Category B at step **360**. If (CURR_BW≧PEAK_BW) or (CURR_BW≧MAX_BW), then the QoS queue is associated with Category C at step **370**.

At step **380**, if all of the QoS queues have been evaluated, then processing branches to step **310**; otherwise, processing continues with step **330**.

Scheduling Processing

Briefly, at each port, three levels of arbitration may be employed to select the appropriate QoS queue from which to transmit the next packet. The first level of arbitration selects among the QoS categories. Category A is given priority if any member QoS queues have one or more pending packets. Otherwise, a QoS queue with one or more pending packets of Category B is selected. According to one embodiment, the relative priority assigned to each QoS queue may be used as a second level of arbitriation. In this manner, when multiple QoS qLueLies satisfy the first level arbitration, a higher priority QoS queue is favored over a lower priority QoS queue. Finally, when there is a tie at the second level of arbitration (e.g., two or more QoS queues in the same QoS category have the same relative priority), a round robin or least recently used (LRU) scheme may be employed to select from among the two or more QoS queues until the QoS categories are evaluated.

Assuming a periodic evaluation of QoS categories is being performed, the scheduling processing need not include such evaluation and the scheduling processing may be performed as illustrated by FIG. **4**, according to one embodiment of the present invention. In the embodiment depicted, at step **410**, processing loops until the port associated with the group of QoS queues being evaluated indicates it is ready to receive the next packet for transmission. For example, the port may be polled to determine its transmission status. Alternatively, the scheduling process may be triggered by an interrupt. In any event, when the port is ready for the next packet, processing continues with step **420**.

At step **420**, a QoS category is selected from which a QoS queue will provide the next packet for transmission. As described above, priority is given to the category containing QoS queues with pending data that are below the peak bandwidth and minimum bandwidth (e.g., Category A). However, if no QoS queues meet this criteria, Category B is selected.

At step **430**, if multiple QoS queues are members of the selected QoS category, processing continues with step **440**, otherwise, processing branches to step **470**.

At step **440**, the relative priorities of the QoS queues are used to select among the QoS queues of the selected category that have pending data.

At step **450**, it two or more QoS queues have the same priority, then processing continues with step **460**. Otherwise, if a QoS queue is found to have the highest relative priority, then processing branches to step **470**.

At step **460**, the tie is resolved by performing round robin or LRU scheduling. That is, until the QoS categories are evaluated, the QoS queues having the same priority will be rotated through in a predetermined order or scheduled such that the QoS queue that has not provided a packet for transmission recently will be given such an opportunity. After selecting a QoS queue in this manner, processing continues with step **470**.

At step **470**, a packet is dequeued from the selected QoS queue and the packet is transmitted by the port at step **480**. This scheduling process may be repeated by looping back to step **410**, as illustrated.

Queuing Schemes

A variety of different queuing mechanisms may be implemented using various combinations of the QoS profile attributes discussed above. Table 2 below illustrates how to achieve exemplary queuing mechanisms and corresponding configurations of the QoS profile attributes.

TABLE 2

Queuing Mechanism Configurations

Queuing Mechanism	QoS Profile Attribute Value
Strict Priority Queuing	Minimum Bandwidth = 0% Maximum Bandwidth = 100% Peak Bandwidth = 100% Maximum Delay = N/A Relative Priority = $PRIORITY_i$
Round Robin/Least Recently Used Queuing	Minimum Bandwidth = 0% Maximum Bandwidth = 100% Peak Bandwidth = 100% Maximum Delay = N/A Relative Priority = <same for all queues>
Weighted Fair Queuing	Minimum Bandwidth = >0% Maximum Bandwidth = MAX_BW_i Peak Bandwidth = $PEAK\_BW_i$ Maximum Delay = N/A Relative Priority = <same for all queues>

$PRIORITY_i$ represents a programmable priority value for a particular QoS queue, i. Similarly, MAX_BW_i and $PEAK\_BW_i$ represent programmable maximum bandwidths and peak bandwidths, respectively, for a particular QoS queue, i.

For a strict priority scheme, each QoS queue's minimum bandwidth is set to zero percent, each QoS queue's maximum bandwidth is set to one hundred percent, and each QoS queue's peak is set to one hundred percent. In this manner, the current bandwidth will never be less than the minimum bandwidth, and the current bandwidth will never exceed either the peak bandwidth or the maximum bandwidth. In this configuration, all QoS queues will be associated with Category B since no QoS queues will satisfy the criteria of

either Category A or Category B. Ultimately, by configuring the QoS profile attributes in this manner, the second level of arbitration (e.g., the relative priority of the QoS queues) determines which QoS queue is to source the next packet.

For a pure round robin or least recently used (LRU) scheme, the QoS profile attributes are as above, but additionally all QoS queue priorities are set to the same value. In this manner, the third level of arbitration determines which QoS queue is to source the next packet.

Finally, weighted fair queuing can be achieved by assigning, at least, a value greater than zero percent to the desired minimum bandwidth. By assigning a value greater than zero to the minimum bandwidth parameter, the particular QoS queue is assured to get at least that amount of bandwidth on average because the QoS queue will be associated with Category A until at least its minimum bandwidth is satisfied. Additionally, different combinations of values may be assigned to the peak and maximum bandwidths to prevent a particular QoS queue from monopolizing the link.

Alternative Embodiments

While evaluation of QoS categories has been described above as occurring periodically, this evaluation may also be triggered by the occurrence of a predetermined event. Alternatively, evaluation of QoS categories may take place as part of the scheduling processing rather than as part of a separate periodic background process.

While a relationship between the number of priority levels and the number of QoS queues has been suggested above, it is appreciated that the number of QoS queues may be determined independently of the number of priority levels. Further, it is appreciated that the number of QoS queues provided at each port may be fixed for every port or alternatively a variable number of QoS queues may be provided for each port.

Finally, in alternative embodiments, weighting factors and ratios other than those suggested herein may be used to adjust the current bandwidth calculation for a particular implementation.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method for bandwidth management in a packet forwarding device, comprising:

identifying a quality of service (QoS) metric corresponding to a traffic group, the QoS metric defining a minimum QoS for the traffic group;

receiving a data packet associated with the traffic group;

placing the data packet into one of a plurality of queues;

identifying a current measure of network performance with respect to parameters specified in the QoS metric; and

removing the data packet from the queue if a difference between the current measure and the minimum QoS falls within a threshold.

2. The method of claim 1 wherein identifying the QoS metric corresponding to a traffic group further comprises:

identifying the traffic group through an Internet Protocol (IP) subnet membership identifier; and

determining a corresponding QoS metric defining a minimum QoS for the traffic group.

3. The method of claim 1 wherein identifying the QoS metric corresponding to a traffic group further comprises:

identifying the traffic group through a media access control (MAC) address; and

determining a corresponding QoS metric defining a minimum QoS for the traffic group.

4. The method of claim 1 wherein identifying the QoS metric corresponding to a traffic group further comprises:

identifying the traffic group through a virtual local area network (VLAN) identifier; and

determining a corresponding QoS metric defining a minimum QoS for the traffic group.

5. The method of claim 1 wherein identifying the QoS metric comprises receiving information indicating a minimum bandwidth for the traffic group.

6. The method of claim 1 wherein identifying the QoS metric comprises receiving information indicating a maximum sustained bandwidth for the traffic group.

7. The method of claim 6 wherein identifying the QoS metric comprises receiving information indicating a peak bandwidth representing a bandwidth in excess of the maximum sustained bandwidth that the traffic group can utilize.

8. The method of claim 1 wherein identifying the QoS metric comprises receiving information indicating a maximum allowable delay for the traffic group.

9. The method of claim 1 wherein identifying the QoS metric comprises receiving information indicating a relative priority associated with the traffic group.

10. The method of claim 1 wherein determining a current measure of network performance occurs at specified intervals of time.

11. The method of claim 1 wherein determining a current measure of network performance with respect to parameters specified in the QoS metric comprises calculating the current measure for the parameters specified in the QoS metric.

12. The method of claim 1 wherein receiving the data packet comprises receiving the data packet on a first port of a plurality of ports, and wherein removing the data packet from the queue comprises transmitting the data packet from a second port of the plurality of ports.

13. The method of claim 1 wherein the packet forwarding device employs a non-deterministic access protocol.

14. The method of claim 13 wherein the non-determninistic access protocol employed by the packet forwarding device is the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol.

15. An article of manufacture comprising a machine accessible medium having content that when accessed provides instructions to cause an electronic system to:

identify a quality of service (QoS) metric corresponding to a traffic group, the QoS metric defining a minimum QoS for the traffic group;

receive a data packet associated with the traffic group;

place the data packet into one of a plurality of queues;

identify a current measure of network performance with respect to parameters specified in the QoS metric; and

remove the data packet from the queue if a difference between the current measure and the minimum QoS falls within a threshold.

16. The article of manufacture of claim 15 wherein the content to provide instructions to cause the electronic system to identify the QoS metric corresponding to a traffic group further comprises the content to provide instructions to cause the electronic system to:

identify the traffic group through an Internet Protocol (IP) subnet membership identifier; and

determine a corresponding QoS metric defining a minimum QoS for the traffic group.

17. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to identify the QoS metric corresponding to a traffic group further comprises the content to provide instructions to cause the electronic system to:

identify the traffic group through a media access control (MAC) address; and

determine a corresponding QoS metric defining a minimum QoS for the traffic group.

18. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to identify the QoS metric corresponding to a traffic group further comprises the content to provide instructions to cause the electronic system to:

identify the traffic group through a virtual local area network (VLAN) identifier; and

determine a corresponding QoS metric defining a minimum QoS for the traffic group.

19. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to identify the QoS metric comprises the content to provide instructions to cause the electronic system to receive information indicating a minimum bandwidth for the traffic group.

20. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to identify the QoS metric comprises the content to provide instructions to cause the electronic system to receive information indicating a maximum sustained bandwidth for the traffic group.

21. The article of manufacture of claim **20** wherein the content to provide instructions to cause the electronic system to identify the QoS metric comprises the content to provide instructions to cause the electronic system to receive information indicating a peak bandwidth representing a bandwidth in excess of the maximum sustained bandwidth that the traffic group can utilize.

22. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to identify the QoS metric comprises the content to provide instructions to cause the electronic system to receive information indicating a maximum allowable delay for the traffic group.

23. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to identify the QoS metric comprises the content to provide instructions to cause the electronic system to receive information indicating a relative priority associated with the traffic group.

24. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to determine a current measure of network performance comprises the content to provide instructions to cause the electronic system to determine the current measure at specified intervals of time.

25. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to determine a current measure of network performance with respect to parameters specified in the QoS metric comprises the content to provide instructions to cause the electronic system to calculate the current measure for the parameters specified in the QoS metric.

26. The article of manufacture of claim **15** wherein the content to provide instructions to cause the electronic system to receive the QoS metric comprises the content to provide instructions to cause the electronic system to receive the data packet on a first port of a plurality of ports, and wherein the content to provide instructions to cause the electronic system to remove the data packet from the queue comprises the content to provide instructions to cause the electronic system to transmit the data packet from a second port of the plurality of ports.

27. The article of manufacture of claim **15** wherein the packet forwarding device employs a non-deterministic access protocol.

28. The article of manufacture of claim **27** wherein the non-deterministic access protocol employed by the packet forwarding device is the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol.

* * * * *

removing adding

US006625150B1

(12) **United States Patent**
Yu

(10) **Patent No.: US 6,625,150 B1**
(45) **Date of Patent: Sep. 23, 2003**

(54) **POLICY ENGINE ARCHITECTURE**

(75) Inventor: **JungJi John Yu**, Saratoga, CA (US)

(73) Assignee: **Watchguard Technologies, Inc.**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/464,939**

(22) Filed: **Dec. 16, 1999**

Related U.S. Application Data

(60) Provisional application No. 60/112,859, filed on Dec. 17, 1998.

(51) **Int. Cl.**[7] **H04L 12/28**

(52) **U.S. Cl.** **370/389**; 370/230; 370/235; 709/238

(58) **Field of Search** 370/229, 230, 370/230.1, 231, 235, 389, 392, 400, 401, 419, 428, 429; 709/238; 713/153, 154

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,371,852 A		12/1994	Attanasio et al.	
5,473,599 A		12/1995	Li et al.	
6,041,053 A	*	3/2000	Douceur et al.	370/389
6,078,953 A	*	6/2000	Vaid et al.	709/223
6,104,700 A	*	8/2000	Haddock et al.	370/235
6,157,955 A	*	12/2000	Narad et al.	709/228
6,292,465 B1	*	9/2001	Vaid et al.	370/230

OTHER PUBLICATIONS

U.S. patent application Ser. No. 09/465,123, Lin, filed Dec. 16, 1999.

Adadi, M., et al, "Secure Web Tunneling," http://pa-.bell-labs.com/~abadi/Papers/tunnel/206.html, pp. 1–13 (Dec. 16, 2000).

"Intel ISP Program Case Studies: UUNET Canada Leads the Industry in Move to Virtual Private Networks," http://ww-w.intel.com/isp/casestudies/uunet.htm, pp. 1–4 (2000).

"Tunnel Switching: 3Com Technology Boosts VPN Security and Flexibility," http://www.3com.com/technology/tech_net/white_papers/503049.html, pp. 10 (1999).

"Virtual Multi-megabit Access Path: Affordable and Available Internet and IP Access at Speeds Greater than T1," http://www.tiaranetworks.com/vmapwp.html, pp. 1–9 (1999).

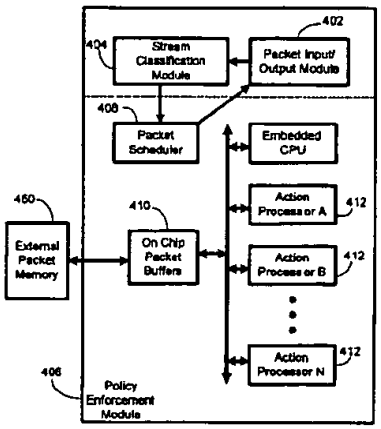
(List continued on next page.)

Primary Examiner—Kenneth Vanderpuye
Assistant Examiner—Bob A. Phunkulh
(74) *Attorney, Agent, or Firm*—Arnall Golden Gregory LLP

(57) **ABSTRACT**

A policy engine for handling incoming data packets. The policy engine includes a stream classification module, a data packet input/output module, and a policy enforcement module. The policy enforcement module further includes a packet scheduler, an on-chip packet buffer circuitry, and a plurality of action processors. The stream classification module creates a packet service header for each data packet, wherein the packet service header indicates policies to be enforced for that data packet. The action processors enforce the policies.

1 Claim, 5 Drawing Sheets



Policy Engine Architecture

OTHER PUBLICATIONS

"Web Workshop—Virtual Private Networking: An Overview," http://msdn.Microsoft.com/workshop/server/feature/vpnovw.asp, pp. 1–16 (May 29, 1998).

Ferguson, Paul and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," http://andrew2.Andrew.cmu.edu/rfc/rfc2267.html, pp. 1–16 (Jan. 1998).

"The NetBoost Policy Engine: Comprehensive Platform Enables Today's Leading Policy Enforcement Applications to Operate at Full Wire Speed," NetBoost Corporation, pp. 1–9 (1998).

"The NetBoost Policy Appliance: Device Enables Concurrent Operation of Multiple Policy Enforcement Applications to Operate at Full Wire Speed," NetBoost Corporation (1998).

"NetBoost PE—1000: Network Application Engine," NetBoost Corporation (1998).

NetBoost SKD: Software Development Kit, NetBoost Corporation (1998).

"A New Breed: The Net Boost Platform for Policy Enforcement Applications," NetBoost Corporation, pp. 1–11 (1998).

Russell, Paul, "Keeping the TCP/IP Stream Flowing," Linux Magazine, http://www.linux-mag.com/1999-08/bestdefemse0.2html, pp. 1–8 (Aug. 1999).

* cited by examiner

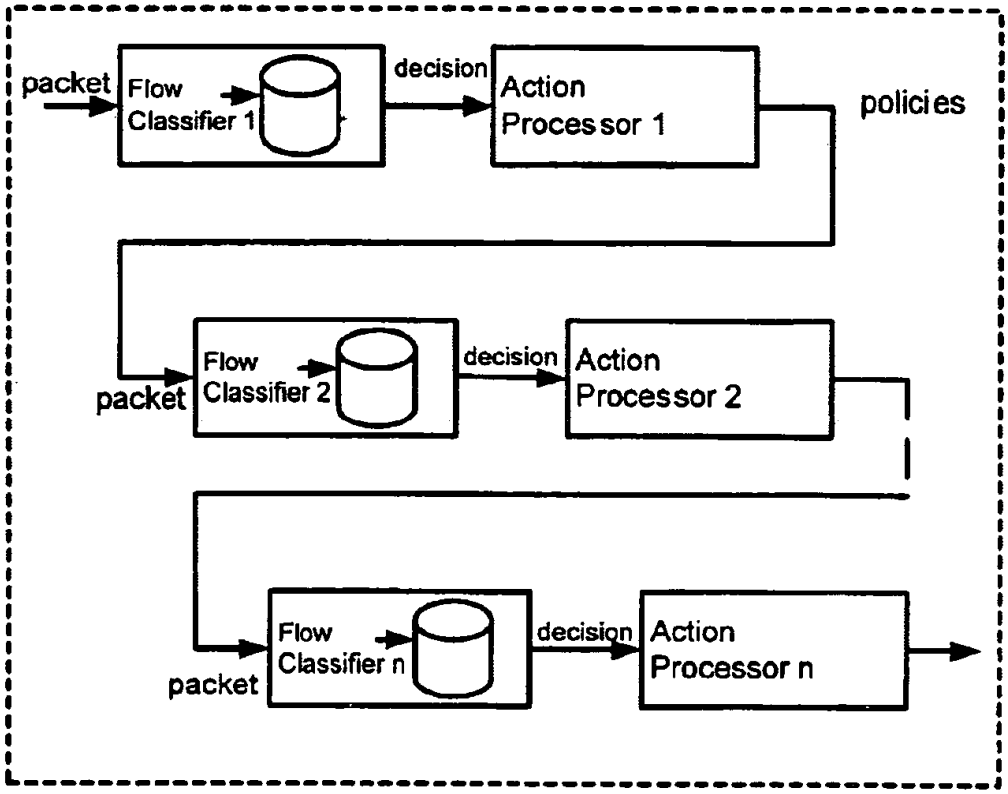


Fig. 1
(prior art)

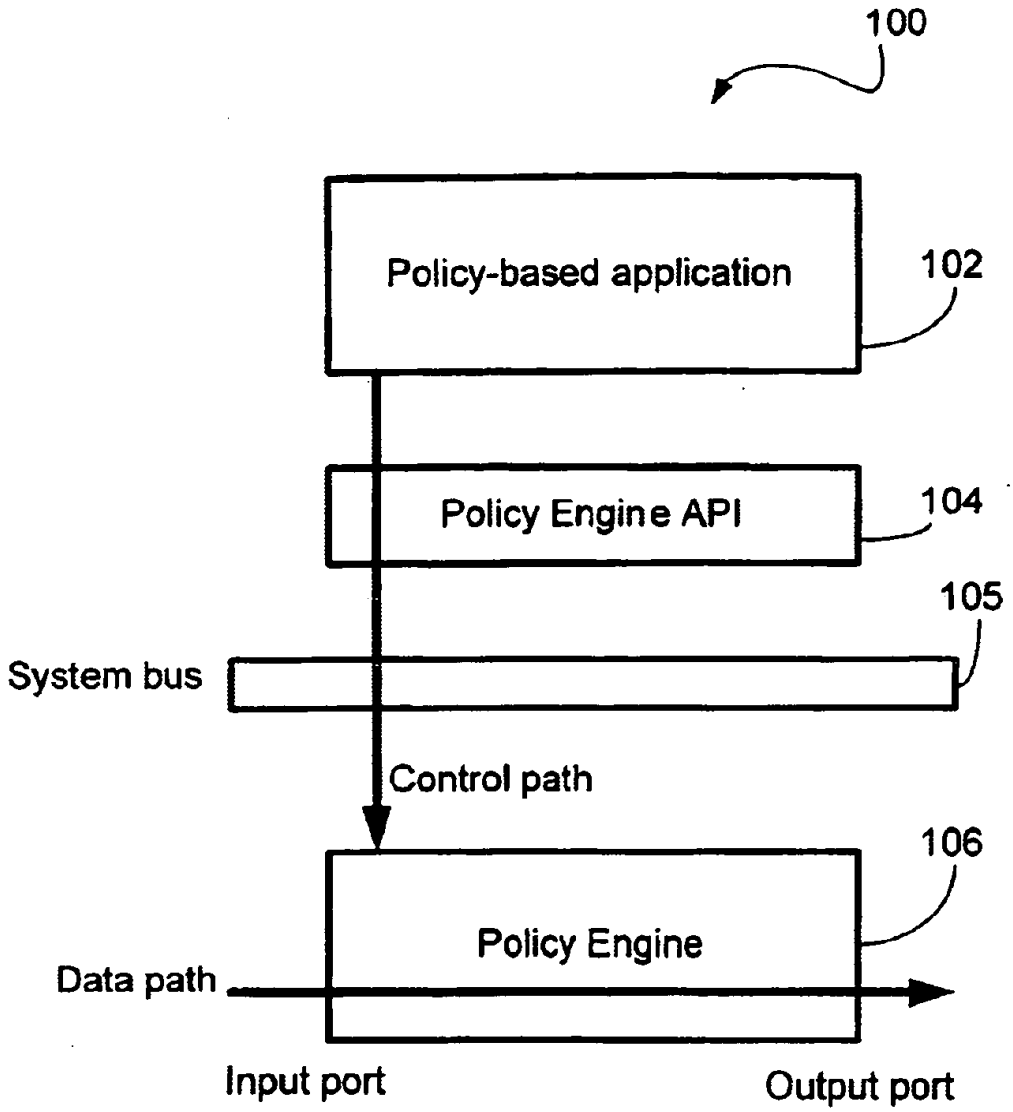

100
Policy-based application
102
Policy Engine API
104
105
System bus
Control path
106
Policy Engine
Data path
Input port
Output port

Fig. 2

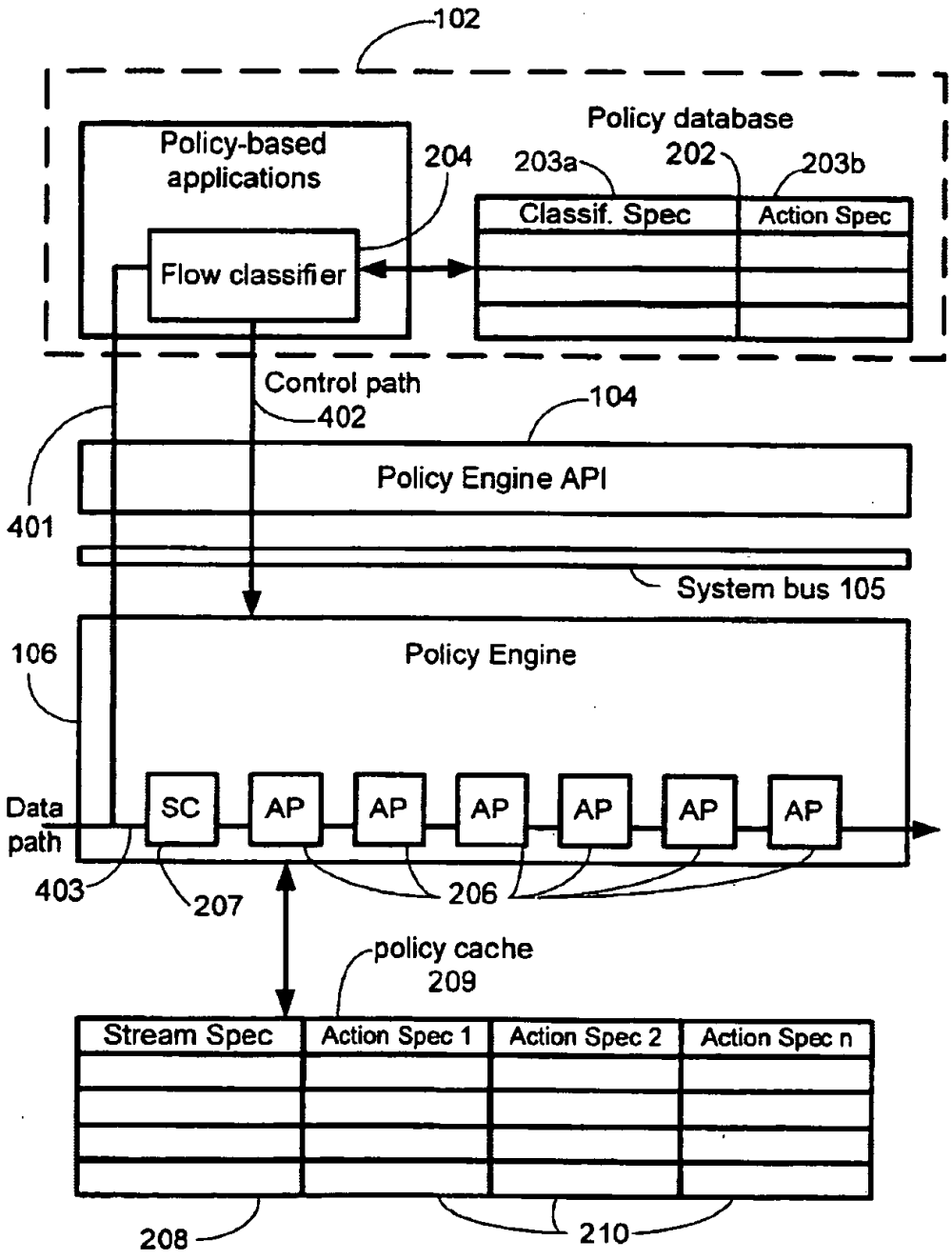

102
Policy-based applications
204
Policy database
203a
202
203b
Classif. Spec
Action Spec
Flow classifier
Control path
402
104
Policy Engine API
401
System bus 105
106
Policy Engine
Data path
SC
AP
AP
AP
AP
AP
AP
403
207
206
policy cache
209
Stream Spec
Action Spec 1
Action Spec 2
Action Spec n
208
210

Fig. 3

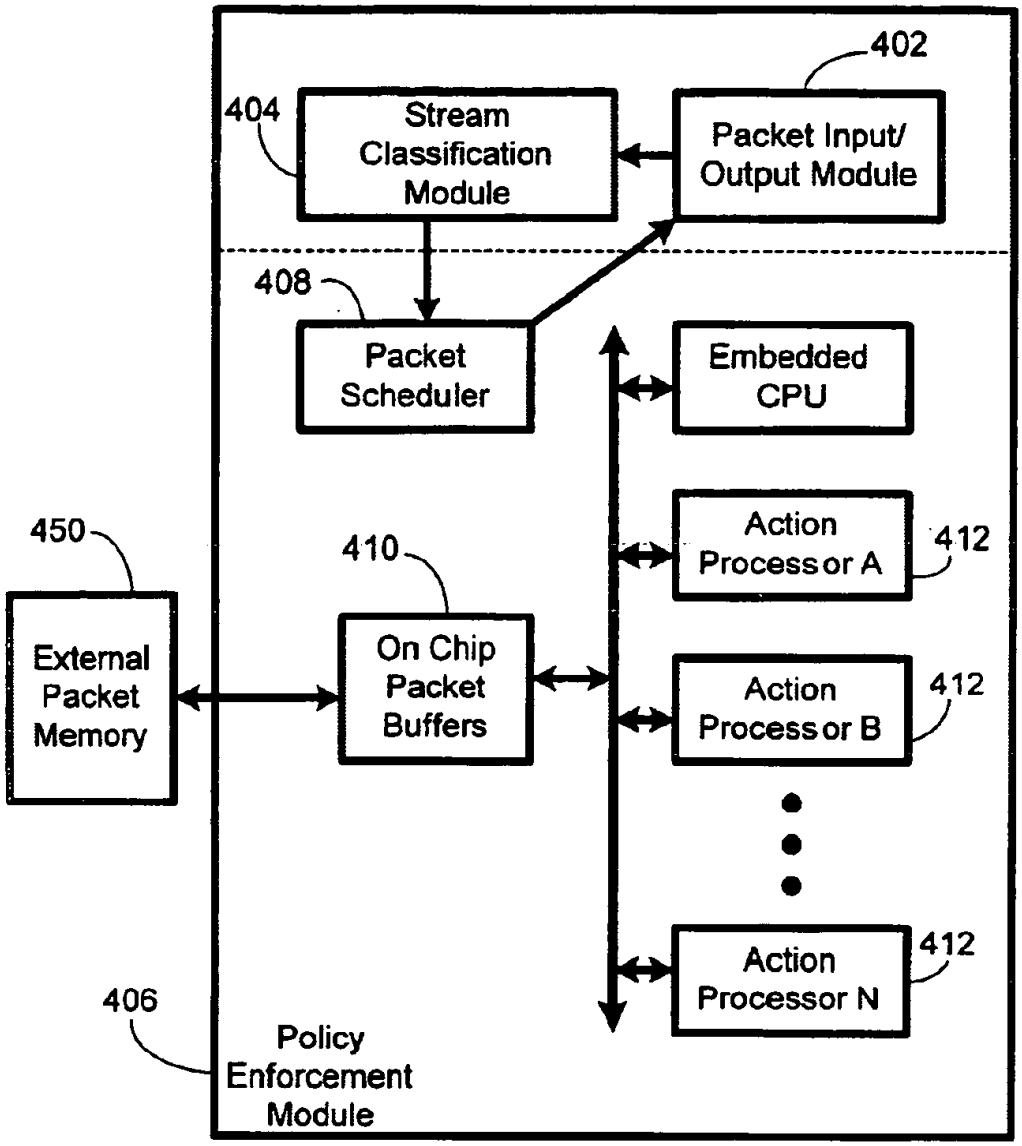


Fig. 4
Policy Engine Architecture

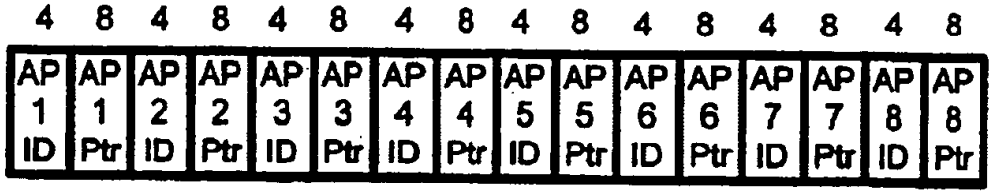


Fig. 5
12-byte Packet Service Header

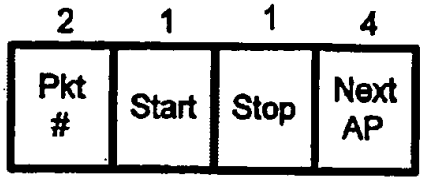


Fig. 6
8-bit Cell Service Header

Packet Service Header: 12-byte Packet Service Header is stored in the On Chip Packet Buffers. One Packet Service Header is generated per incoming packet.

AP x ID: the # x Action Processor ID.
AP x Ptr: A pointer to Action Processor's action spec (e.g. DES Key) of the # x.

Cell Service Header: For the purpose of creating deep pipeline to maximize the policy engine performance. Packets are fragmented into fixed-size cells. One-byte Cell Service header is attached to each cell and travel along with cells to the Action Processors.

Pkt#: Relative Packet number to ensure packet ordering. Pkt# is given when copied to On Chip Packet Buffer, 00 > 01 > 11 > 00

Start: First cell of a packet.
Stop: Last cell of a packet. If both Start and Stop bit cell, packet size is equal to or smaller than a cell (64B).
Next AP: Next Action Processor to perform on the cell. "0000" indicates completion of all action processing.
Hop P licy Enforcement Unit.

POLICY ENGINE ARCHITECTURE

RELATED APPLICATIONS

This application claims the benefit of priority to U.S. Provisional Patent Application Ser. No. 60/112,859, filed Dec. 17, 1998.

TECHNICAL FIELD

The present invention relates to policy-based network equipment and, in particular, to policy-based network equipment that employs a favorable division of hardware and software to provide both performance and flexibility.

BACKGROUND

Some typical policy-based computer network applications are Virtual Private Networks (VPN), Firewall, Traffic Management, Network Address Translation, Network Monitoring, and TOS Marking. In general, the policy-based application has access to the network media through an operating system driver interface. In a typical network architecture, the policy-based application examines every packet coming in from the network along the data path, compares it against flow classification criteria, and performs the necessary actions based upon the policies defined in a policy database.

Today's policy-based applications are challenged with several key issues. These issues can be major inhibitors for the future growth of the emerging industry:

1) Flow classification overhead—Flow classification specifications can be complicated and lengthy for each network service. As can be seen from FIG. **1**, in a conventional policy-based application, each packet compared with potentially hundreds of rules in order to find the matching one and determine the proper action specifications. With stateful applications, state tracking is even more time consuming. Multiple network services on a single system simply make matters worse.

   As is also shown in FIG. **1**, the process of flow classification and action processing may repeat for many iterations as multiple policies are activated at the same time. For example, a VPN (virtual private network) application may comprise Firewall Policy, IPSEC Policy, IPCOMP (IP compression) policy, NAT (Network Address Translation) Policy, QoS (Quality of Service) policy, Monitoring Policy, L2TP/PPTP (L2 Tunnel Protocol/Point To Point Tunnel Protocol) Tunnel Policy, and so on.

   The flow classification is a rule based operation that can be very flexible to tune to application needs. For example, it may define a rule to identify packets with a pattern of any random byte within a packet, and/or across many packets. The flow classifiers may also differ per action processor for performance optimization. As a result the matching criteria used by a flow classifier to classify a flow may include a specific value, a range, or wildcard on interface port numbers, protocols, IP addresses, TCP ports, applications, application data, or any user specifiable criteria. The distinctions of various implementation makes it difficult to cache a flow with its decision in many ways.

2) Flow classification technique is evolving—Flow classification and analysis technique is more than just looking into the packet's address, port number and protocol type and or other header information. It often involves state tracking for newer applications. This technique is being continuously modified and, therefore, is not practically appropriate for a hardware based implementation. Furthermore, flow classification techniques are often viewed as key differentiaters between vendors.

3) Action execution speed—Once the classification process is complete, the proper actions need to be executed. Some of the actions are simple like a discard or forwarding decision for a firewall, while some others are extremely time consuming, like triple-DES encryption and SHA hashing algorithm or QOS scheduling algorithm. Software based implementations cannot keep up with the bandwidth expansion as newer and faster media technologies are employed.

4) Integrated services—As more and more policy-based applications become available, it is desirable to provide integrated services on a single platform because this ostensibly reduces policy management complexity, avoids potential policy conflicts, and lowers the TCO (Total Cost of Ownership). On the other hand, integrated services impose a very large computing power requirement that cannot be practically achieved with off-the-shelf general purpose machines. A disadvantage of the conventional architecture is that, because it is primarily software-based, it is relatively high overhead. However, precisely because it is software-based, it is quite flexible.

What is desired is a policy architecture has the flexibility of present flow classification systems, but that also has lower overhead.

BRIEF DESCRIPTION OF THE FIGURES

FIG. **1** is a block diagram illustrating conventional flow classification and action processing.

FIG. **2** is a block diagram illustrating the a broad aspect of a policy architecture in accordance with an embodiment of the invention.

FIG. **3** is a block diagram illustrating details in accordance with one embodiment of FIG. **2**.

FIG. **4** illustrates more details of how the FIG. **3** architecture is employed to process network data.

FIGS. **5** and **6** illustrate headers added to the packets by the stream classification module for use by the action processors.

DETAILED DESCRIPTION

As shown broadly in FIG. **2** and in greater detail in FIG. **3**, in accordance with one embodiment of the invention, an architecture **100** for applying policies to network data traffic allocates the application of policies between software and hardware such that the system is flexible yet efficient.

The architecture **100** includes three major components—a Policy-Based Application **102**, a Policy engine API **104** ("API" stands for Application Program Interface") and a Policy engine **106**. As can be seen from FIGS. **2** and **3**, the policy-based application **102**—such as a firewall, virtual private network (VPN), or traffic management—is typically a "legacy" software program residing on a host, equipped with its own policy database **202** and flow classifier logic **204**.

The policy engine API **104** serves as an interface between the policy application **102** and the policy engine **106** (via a system bus **105**). The policy engine **106** is a purpose-built hardware (preferably running at wire speed) that operates on input network traffic and network policies and that outputs regulated traffic flows based upon the network policies.

In a typical embodiment, the policy engine API **104** provides the policy-based application **102** access to all the

media I/O through a generic operating system driver interface. In addition, the API **104** allows the application **104** to invoke acceleration functions (shown in FIG. **3** as application processors **206**, or "AP's") provided by the policy engine **106**. The application processors **206** operate based on the stream classifier **207** of the policy engine **106** determining that a packet belongs to a particular stream and activating the appropriate action processors **206** according to action specifications **210** in a policy cache **209**. That is, overall system performance is enhanced by virtue of the appropriate acceleration functions (action processors **206**) of the policy engine **106** being activated to regulate the network traffic.

Before proceeding, several terms are defined in the context of FIGS. **2** and **3**. The definitions provided herein are meant to be explanatory, and not necessarily limiting when a similar or identical term is used in the claims.

Service

A service in a policy-based network defines a network application **102** that is controlled and managed based on a set of policies. Typical services are firewall, VPN, traffic management, network address translation, network monitoring, etc.

Policy

Policies (normally defined by network managers) are collectively stored in a policy database **202** accessible to the policy-based applications **102** (even conventionally) and describe network traffic behaviors based upon business needs. A policy specifies both what traffic is to be subject to control and how the traffic is to be controlled. Thus, a policy typically has two components—a flow classification specification **203***a* and an action specification **203***b*.

Flow Classification Specification **203***a*

A flow classification specification **203***a* provides the screening criteria for the flow classifier logic **204** to sort network traffic into flows. A flow classification specification **203***a* can be very elaborate, as detailed as defining a specific pair of hosts running a specific application. Alternately, a flow classification specification **203**a can have a simple wildcard expression.

Action Specification **203***b*

An action specification **203***b* describes what to do with packets that match an associated flow classification specification **203***a*. The action specification **203***b* can be as simple, for example, as a discard or forward decision in the firewall case. It can also be as complicated as IPSec encryption rules based on a SA (Security Association) specification.

Flow

All packets that match the same flow classification specification **203***a* form a flow.

Flow Classifier

Referring again to FIG. **3**, a policy decision is at least initially derived by a policy-based application from the policy database **202**. As discussed above, a flow is a stream of correlated packets to which policy decisions apply. With the described embodiments in accordance with the invention, referring again specifically to FIG. **3**, for at least some of the packets, a flow classifier **204** classifies the packet according to one or more classification specifications **203***a* and finds one or more corresponding action specifications **203***b*. The found action specifications **203***b* are then provided to the policy cache **209** for later execution by the policy engine **106** to enforce the policy.

Policy Binding

Policy binding is the process of the flow classifier **204** binding a stream with its associated action specification and loading the appropriate entries (stream specification **208** and action specifications **210**) into the policy cache **209**.

Stream

A stream is an "instantiation" of a flow—packets that have the same source and destination address, source and destination port, and protocol type. (Optionally, the application can add the input and output media interface to the stream classification criteria in addition to the packet header if desired.) Packets may be sorted into streams, and a flow may include one or more streams. All packets belonging to the same stream are to be regulated by the same policy.

Policy Cache **209**

At the completion of the policy binding process, an entry for a given stream is created on the policy engine which contains all the policy information required to subsequently process data of the stream

Integrated Services

When multiple network services are to apply to the same flow, this is called "Integrated Services". Integrated Services simplify the management of various service policies, minimize potential policy conflicts and reduce TCO (Total Cost of Ownership).

Stream Specification

A stream specification **208**, shown in FIG. **3** held in policy cache **208** is the criteria used by the stream classifier **207** to uniquely identify a stream. In one embodiment, the stream specification **208** is compared to a 5-tuple in a packet header—source and destination address, source and destination port, and protocol type.

Action Processor **206**

Each action processor **206** executes an action based upon an action specification **210** in the policy cache **209**.

Packet Tagging

Certain applications (e.g. Network Monitoring) would like to receive flows based on the flow classification specification and would prefer that flow classification be performed for them. Packet tagging is a way of tagging all incoming packets with an application specified "tag

Policy-based Application

A policy-based application provides a service to the network users. This service is managed by a set of policies. Firewall, VPN and Traffic Management are the most typical policy-based applications. As the industry evolves, policy-based applications are likely to consolidate onto a single platform called Integrated Services. Integrated Services has the benefits of centralized policy management and lower cost of ownership.

Referring still to FIG. **3**, the population and use of the policy cache **209** is now discussed in greater detail. As discussed above, the policy-based application **102** (typically a legacy application) is equipped with its own policy database **202** and flow classifier logic **204**. Some of the packets of a stream are provided (via a data path shown logically as **401** in FIG. **3**) to the flow classifier **204**. The flow classifier **204** uses the policy database **202** to determine the action specifications **203***b* that correspond to the policies of the flow to which the stream belongs. The action specifications are provided (via the path shown logically as **402** in FIG. **3**) to the policy cache **209**. It should be noted that multiple packets may be required for more sophisticated flow classification (stateful packet inspection), since the policy decisions (action specifications) may come from different applications which may have implemented different flow classifiers. In those cases, the application's flow classification logic keeps track of the flow's state until a matching criteria is met. Preferably, though, just enough packets of a stream are provided to the flow classification logic **204** via theological path **401** to properly determine the action specifications **203***b* for the stream. At the end of the "learning

phase", the application software **102** has uniquely identified a policy for the incoming packet stream

Subsequent packets of the stream are then provided directly to the stream classifier **207** of the policy engine **106** via the logical data path **403**. Using the policy cache **209**, the stream classifier **207** determines which action processors **206** are to be activated for the packets of the stream. Specifically, the stream classifier **207** matches the packets to a particular stream specification **208** and then, using the corresponding action specifications **210**, activates the proper action processors **206**. Significantly, these "subsequent packets" can be acted upon without. any interaction to the "host" policy-based application **102**. The application need not "see" any packets belonging to that stream after the binding (unless the stream is actually destined for the host.). The action processors are specialized in executing specific action specifications, preferably at the wire speed.

Thus, in summary, upon the completion of the policy binding "learning" process, the policy engine **106** may immediately take control of the bound stream and execute the appropriate actions in accordance with the action specifications **210** in the policy cache **209** without any intervention from the "host" (policy-based) application. This method also relieves the policy engine **106** hardware from doing complicated pattern matching because it can simply compute a hash value (or use some other identification function) from the well known fields (which uniquely identify a stream) of the packet to find its corresponding policy decisions (action specifications **210**). The classification need not be done more than once for each packet even though there may be multiple applications. As a result, massive computing power is not required to do the classification on an ongoing basis. A benefit is inexpensive hardware cost for very high performance policy-based applications.

It can be seen that in accordance with the present invention, use of the policy engine and policy cache not only addresses many if not all of the performance considerations discussed above in the Background, but also preserves a great amount of flexibility in setting network policies and the following considerations are taken into account.

1) Time-to-market for application developers—Since time-to-market is a major concern for application vendors, the PAPI design minimizes the development effort required by the application developers in order for the existing applications to take advantages of the policy engine's enhanced performance.
2) Maintain flexibility for developers' value-added—PAPI may allow application developers to enhance or maintain their value-add so that vendors' differentiation is not compromised.
3) Platform for integrated services—PAPI has the model of an integrated services platform in mind. Application developers can, over time, migrate their services into an integrated platform without worrying about the extensibility of the API and the performance penalty.

All of the preceding has been disclosed in a co-pending patent application. The focus of the current application is one aspect of the policy engine itself: namely, the architecture of the policy engine itself from a system level. One challenge of this (or, in fact any) hardware assisted solution is not only maximizing parallelism among enforcement of different policies, but also that the architecture requires flexibility among order of enforcement. This is because the policy enforcement is usually order sensitive. For example, decryption of packets needs to be performed after firewall policy enforcement for incoming traffic to an corporate network. In addition to being order sensitive, it is also a challenge to provide for adding new policies, even when the new policies to be added are not even known at the product release.

FIG. **4** illustrates a policy engine architecture in accordance with one embodiment of the invention. The FIG. **4** Policy engine includes a Packet Input/Output Module **402**, a Stream Classification Module **404**, and a Policy Enforcement Module **406**.

The Packet Input/Output Module **402** receives packets, places the received packets in the external packet memory **450** and notifies the Stream Classification Module **404** of such packets. Upon completion of all policies enforcement, the Packet Input/Output Module **402** transmits the packet from external packet memory **450** to the network.

The Stream Classification Module **404**, based on the policy cache, creates a Packet Service Header (shown in FIG. **5**) for each packet. The Packet Service Header indicates what policies need to be enforced and in what order. The Stream Classification Module **404** is software programmable. As shown in FIG. **5**, the Packet Service Header includes a number of pairs AP ID and AP Pointers. An AP ID uniquely defines an Action Processor, and the AP pointer points to the Action Spec required to enforce such policy. An example of an action processor is a DES engine that needs a 56-bit or 112-bit key to do the encryption or decryption. The policy cache can be modified if network requirements changes. In addition to that, the order of different policy enforcement can also be programmed to achieve different application requirements.

The Policy Enforcement Module **406** includes a Packet Scheduler **408**, On Chip packet Buffer(s) **410**, and at least one Action Processor **412**. The Packet Scheduler **408** copies packets from external packet memory **450** to the On Chip Packet Buffer **410**. After copying the packets to the Packet Buffer **410**, packets are fragmented into 64 bytes cells. An 8-bit Cell Service Header (FIG. **6**) is added to the beginning of each 64-byte cell. he Cell Service Header includes a Packet Number to uniquely identify a packet in the Policy Enforcement Module pipeline and a Start bit and Stop bit to indicate the first and last cell of a packet. A Next AP field, together with the AP IDs in the Packet Service Header, indicates to the Policy Enforcement Module **406** what is the next destination Action Processor of each cell.

It is preferable to have the On Chip Packet Buffer **410** because it allows the Action Processors **412** very low latency and high bandwidth access to the packets as compared with having to access the external Packet Memory **450**. In case the next Action Processor **412** is busy for a cell, the On Chip Packet Buffer **410** serves as temporary storage for that cell. This prevents the blocking of following cells which need to go through this same Action Processor **412**.

Each Action Processor **412** performs a particular policy enforcement. In addition to this, it is capable of reading the required action spec based on the AP pointer on the packet Service Header (FIG. **5**). Each Action Processor may also have its own input and/or output FIFO to buffer the cells.

Each cell is routed to the next Action Processor based on the Packet Service Header (FIG. **5**) and the Cell Service Header (FIG. **6**). The cell routing is preferably distributed to each Action Processor, instead of being centralized to a cell routing unit. This distributed approach allows for adding and removing policies much more easily. Upon completion of all policy enforcement for a particular packet, the packet scheduler **408** copies that packet to external packet memory **450**. The Packet Input/output module **402** is then notified and transmits the packet to the network.

What is claimed is:

1. A policy engine comprising:

a stream classification module;

a packet input/output module that places received packets in an external packet memory and that notifies the stream classification module of the packets in the external packet memory;

wherein the stream classification module creates a packet service header for each packet in the external packet memory indicating, based on a policy cache, policies to be enforced on that packet;

a policy enforcement module to enforce policies on the packets, including

a packet scheduler that fragments each packet into cells and schedules enforcement of the policies on each cell based on the packet service header;

on-chip packet buffer circuitry to temporarily hold the packets during policy enforcement; and

a plurality of action processors, each action processor performing a particular policy enforcement on a cell and routing the cell to a next one of the action processors.

* * * * *

US006463068B1

(12) **United States Patent**
Lin et al.

(10) **Patent No.: US 6,463,068 B1**
(45) **Date of Patent: Oct. 8, 2002**

(54) **ROUTER WITH CLASS OF SERVICE MAPPING**

(75) Inventors: **Arthur Lin**, San Ramon; **Thomas J. Edsall**, Mountain View; **Tauseef Hashmi**, Freemont, all of CA (US)

(73) Assignee: **Cisco Technologies, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/001,478**

(22) Filed: **Dec. 31, 1997**

(51) **Int. Cl.**[7] **H04L 12/56**
(52) **U.S. Cl.** **370/414**; 370/230
(58) **Field of Search** 370/412, 468, 370/429, 414, 417, 418, 230, 428

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,953,341 A * 9/1999 Yamanaka et al. ..... 370/412 X

OTHER PUBLICATIONS

S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", Aug., 1993 IEEE/ACM Transactions on Networking, pp. 1–32.

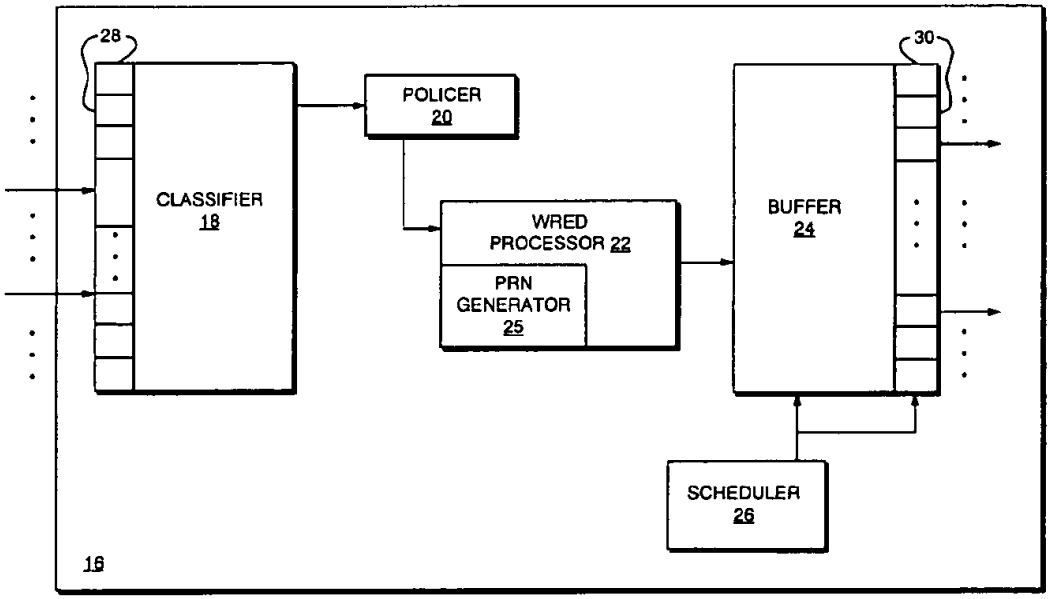
* cited by examiner

Primary Examiner—Joseph L. Felber
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP

(57) **ABSTRACT**

A router includes a classifier that classifies packets assigned to 2^{n-m} classes of service into 2^n classes of service that are supported by the router. The classifier then sets the loss priorities of the respective packets to one of m levels. The router uses a modified weighted random early detection scheme that is based on probabilities of discard associated with the 2^{n+m} classes of service to determine whether to retain or discard particular packets. The router uses a single buffer to store packets directed to all of the various output ports. The available storage locations in the buffer are linked to a free queue and a weighted average depth of the free queue, is used to determined whether or not to retain a given packet. The router compares the weighted average depth of the free queue to maximum and minimum thresholds associated with the particular 2^{n+m} class of service to which the packet is assigned. If the weighted average is above the maximum threshold, the packet is retained. If the weighted average is below the minimum threshold, the packet is discarded. If the weighted average is between the two thresholds, a probability of discard that is based on the 2^{n+m} classes of service is calculated and compared to a random value to determine whether or not the packet should be retained. If the probability of discard exceeds the random value, the packet is discarded. The other packets exit various output ports of the router based on weighting factors associated with the 2^n classes of service.

34 Claims, 5 Drawing Sheets



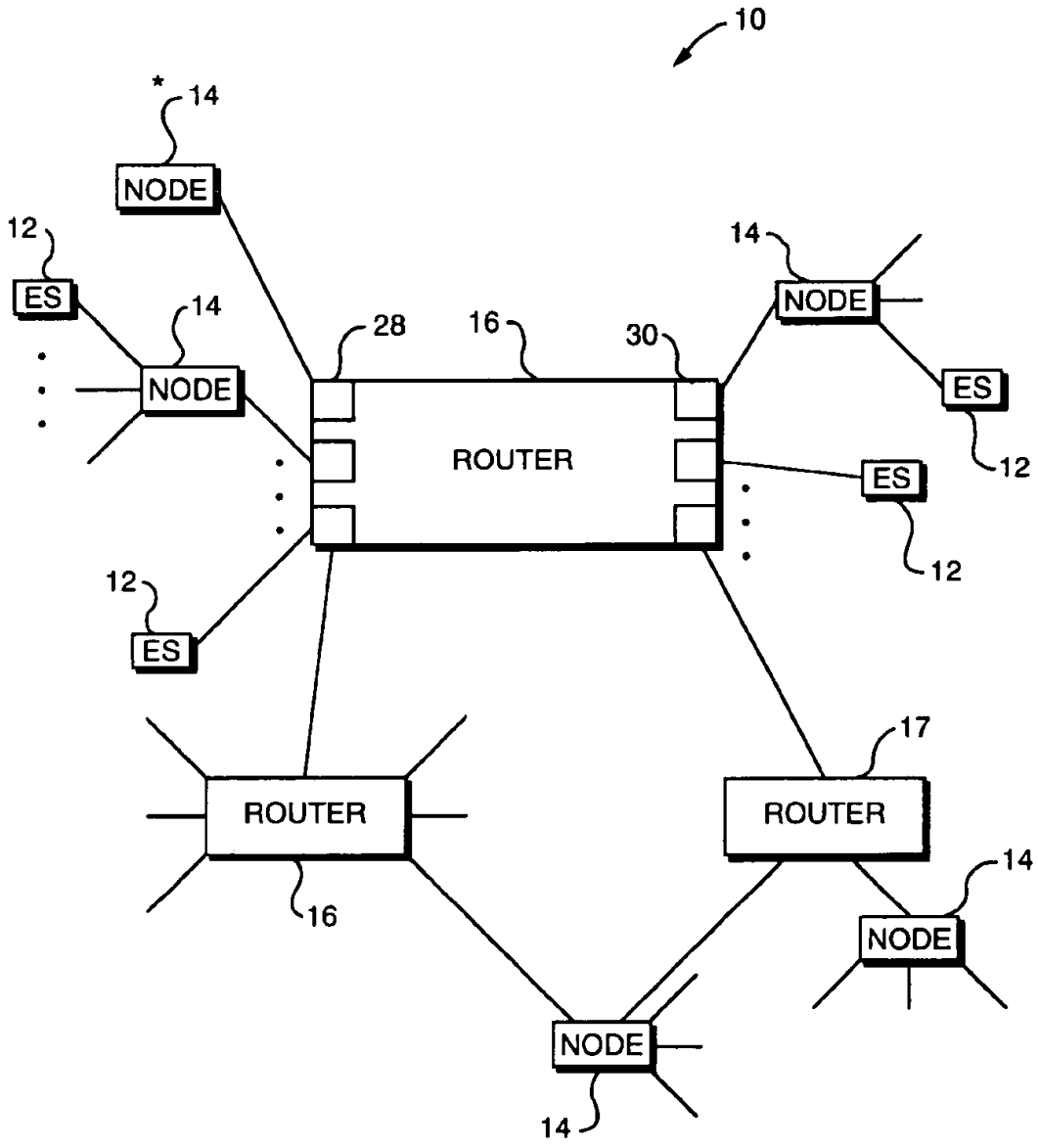

10
14
NODE
12
ES
14
NODE
28
16
30
14
NODE
ES
12
ROUTER
ES
12
12
ES
17
ROUTER
ROUTER
16
14
NODE
NODE
14

FIG. 1

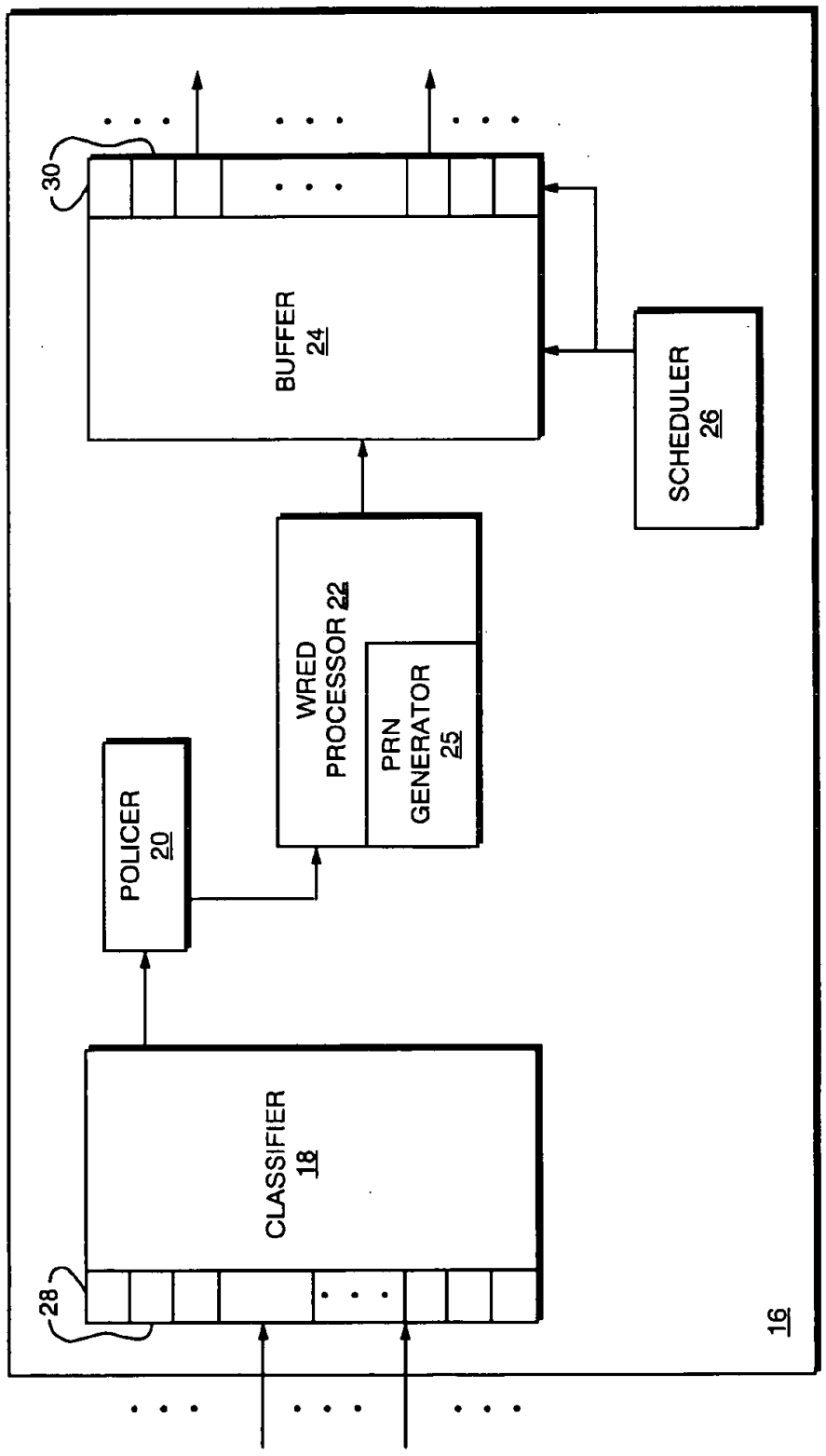

30
BUFFER
24
SCHEDULER
26
WRED
PROCESSOR 22
PRN
GENERATOR
25
POLICER
20
CLASSIFIER
18
28
16

FIG. 2

CLASSES OF SERVICE	COS TAGS	CLASS OF SERVICE PER OUTPUT PORT QUEUES
IS/ISSLL GUARANTEED	000/001	Q_0
ROVING MANAGEMENT	000/001	Q_0
IS/ISSLL CONTROLLED LOAD	010/011	Q_1
RESERVED	100/101	Q_2
BEST EFFORT	110/111	Q_3

FIG. 3

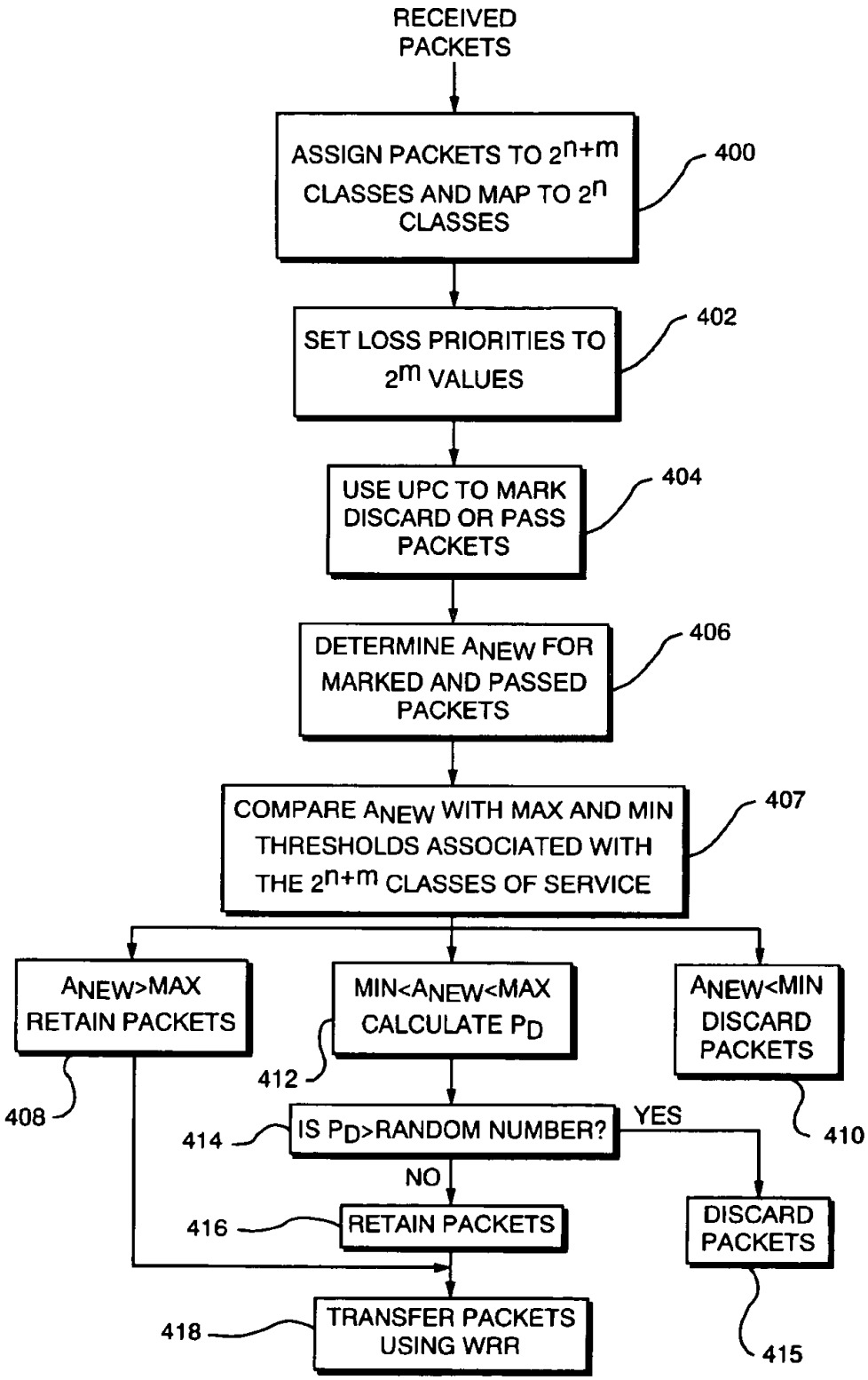

RECEIVED PACKETS
ASSIGN PACKETS TO 2^{n+m} CLASSES AND MAP TO 2^n CLASSES
400
SET LOSS PRIORITIES TO 2^m VALUES
402
USE UPC TO MARK DISCARD OR PASS PACKETS
404
DETERMINE A_{NEW} FOR MARKED AND PASSED PACKETS
406
COMPARE A_{NEW} WITH MAX AND MIN THRESHOLDS ASSOCIATED WITH THE 2^{n+m} CLASSES OF SERVICE
407
A_{NEW}>MAX RETAIN PACKETS
408
MIN<A_{NEW}<MAX CALCULATE P_D
412
A_{NEW}<MIN DISCARD PACKETS
410
414
IS P_D>RANDOM NUMBER?
YES
NO
416
RETAIN PACKETS
DISCARD PACKETS
415
418
TRANSFER PACKETS USING WRR

FIG. 4

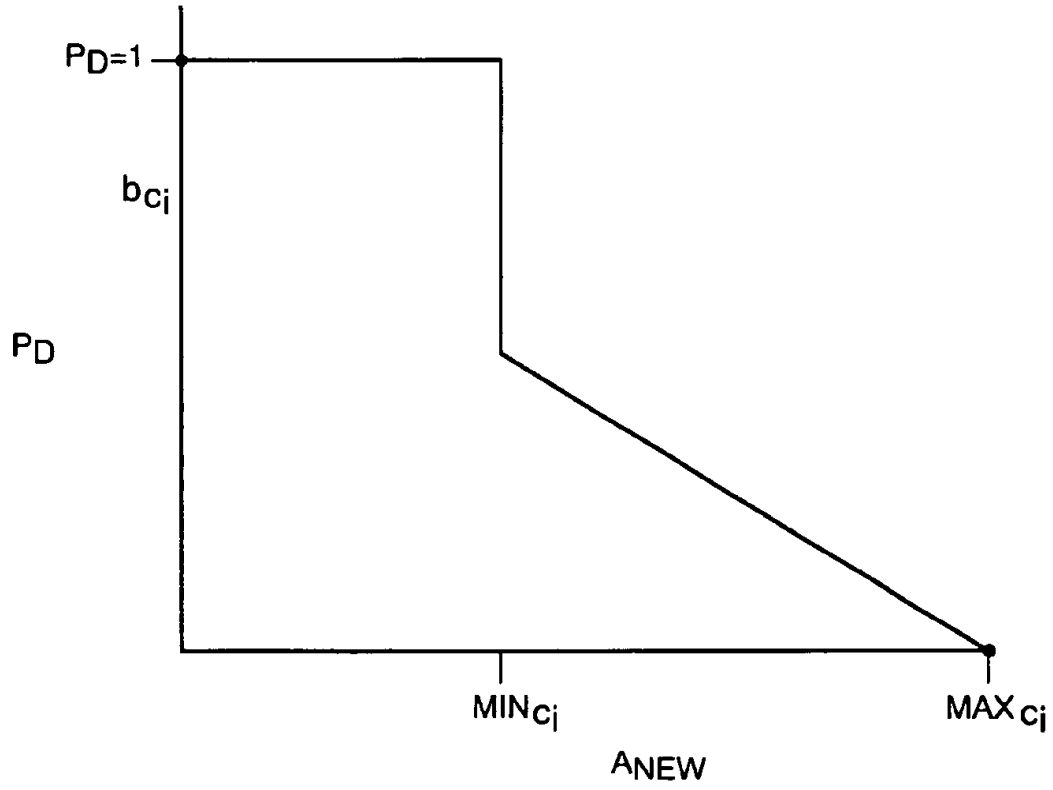


FIG. 5

ROUTER WITH CLASS OF SERVICE MAPPING

FIELD OF INVENTION

The invention relates generally to routers and switches and, more particularly, to routers and switches that support multiple classes of service for packet routing.

BACKGROUND OF THE INVENTION

At network multiplexing points, such as switches or routers, the handling of frames or packets is generally determined by rules associated with classes of service to which given frames or packets are assigned. The classes of service essentially define acceptable packet or frame delays and probabilities of packet or frame loss. (As used herein, the term "packet" refers to both frames and packets, and the term "router" refers to both switches and routers.)

The packets are typically assigned to classes of service based on information contained in the packet and/or traffic management rules established by either a network supervisor or a service provider. All packets assigned to the same class receive the same treatment. Being assigned to a "higher" class ensures that a packet will have a shorter maximum transmission delay and a lower probability of loss. Being assigned to a "lower" class may mean a longer delay and/or a greater probability of loss.

Generally, the router maintains at each output port a buffer for holding packets in queues associated with the classes of service. The queues ensure that packets are delivered in order within the various classes of service, and that the associated rules for maximum delays and probabilities of loss can be enforced. Since each queue is essentially separately maintained, the more classes the router supports the more processing and storage capacity is required for a given number of output ports. To support "x" classes, for example, the router must set aside buffer storage locations for each of the x queues at each of its "y" ports. Further, it must determine for each queue whether or not a next packet should be retained or discarded. The router thus makes x*y separate calculations based on queue length and/or available associated storage locations to determine whether to retain or discard the packets, where "*" represents multiplication.

Network standards, such as the (revised) 802.1p standard, have relatively recently increased the number of classes of service to eight classes. Routers operating under prior standards support four classes of service, and thus, must be upgraded, for example, with increased storage capacities of the output port buffers, to support the increased number of classes. Such upgrading may be prohibitively expensive and/or it may not be feasible. Accordingly, what is needed is a mechanism to operate a router that supports a relatively small number of classes of service in an environment in which packets are assigned to a greater number of classes. Such a mechanism should, without requiring the enlarged storage and processing capabilities conventionally associated with supporting the greater number of classes, maintain service distinctions associated with the greater number of classes and more importantly retain the order of packets within each of the greater number of classes.

SUMMARY OF THE INVENTION

A router maps packets assigned to 2^{n+m} classes of service into 2^n classes of service and assigns the packets to 2^m levels of loss-priority within each of the 2^n classes. The router includes a classifier that uses n bits of an (n+m)-bit "class of service identifier" to map the packets to the 2^n classes, and the remaining m bits to assign the loss priorities. The router then controls packet retention/discard with a modified weighted random early detection scheme based in part on the 2^{n+m} classes and in part on the 2^n classes, to maintain the probability of loss distinctions and in-order packet handling associated with the 2^{n+m} classes.

A scheduler controls the transmission of packets by each output port based on the 2^n classes of service. The scheduler uses a weighted round robin scheme to ensure that packets from each of the classes are transmitted by each of the output ports within the prescribed maximum delay limits associated with the 2^{n+m} classes of service.

The router includes an output buffer that holds the packets for all of the router's output ports. The router maintains a "free queue," which links the buffer storage locations available for packet storage. To determine whether to retain or discard a given packet, the router compares a weighted average depth of the free queue with predetermined maximum and minimum thresholds that are associated with the particular one of the 2^{n+m} classes of service to which the packet is assigned. If the weighted average exceeds the associated maximum threshold, the router retains the packet in a storage location that is then removed from the free queue and linked to a class of service per output port queue that corresponds to the class of service to which the packet is mapped by the classifier. If the weighted average depth falls below the associated minimum threshold, the router discards the packet. If the weighted average depth falls between the associated minimum and maximum thresholds, the router calculates a probability of discard and compares the probability to a "random" value. The router discards the packet if the probability exceeds the random value, and otherwise retains the packet.

The maximum and minimum thresholds are set relative to one another such that the loss priorities associated with the 2^{n+m} classes are maintained. As discussed below, the router makes only one weighted average queue depth calculation for the free queue, and uses this calculation to determine whether to retain or discard packets for the 2^n classes of service. This is in contrast to prior known routers that must maintain at each output port separate average queue depths for each of the class of service per port queues.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention description below refers to the accompanying drawings, of which:

FIG. 1 is a functional block diagram of a network that includes routers that are constructed and operate in accordance with the invention;

FIG. 2 is a functional block diagram of a router of FIG. 1;

FIG. 3 illustrates a mapping of packets to classes of service;

FIG. 4 is a flow chart of the operations of the router of FIG. 2; and

FIG. 5 is a graph of weighted average queue depth versus probability of packet discard.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

Referring now to FIG. 1, a network 10 includes a plurality of endstations 12 and nodes 14 that transmit packets to other endstations 12 and nodes 14 through routers 16 and 17. The

endstations **12** and nodes **14** assign packets to classes of service based on information contained in the packets and/or on predetermined traffic management rules that are provided by the network manager and/or various service providers. The classes of service are essentially associated with maximum limits for transmission delays and probabilities of packet loss. Higher classes are associated with shorter maximum delays and lower probabilities of packet loss. Packets that must be delivered as essential parts of a transmission are, for example, assigned to a higher class than are packets that contain non-essential information.

Preferably, the endstation **12** or the node **14** that introduces the packet to the network assigns the packet to one of 2^{n+m} classes of service. To inform the routers **16** and **17** of the assignment, the endstation **12** or node **14** writes an appropriate class of service (COS) "tag" to a COS identifier field in the header that is included in the packet. The COS identifier field has three bits, as defined by (revised) standard 802.1p, and the packet is thus assigned to one of eight classes of service, i.e., 1 of 2^3 classes. The packet is then forwarded by the endstation **12** or node **14** over the network **10** to an input port **28** of a router **16** or **17**. The router then transfers the packet through an output port **30** and over the network in accordance with the transmission rules and delay limits associated with the class of service to which the packet is assigned.

The routers **17** support 2^{n+m} classes of service while the routers **16** support 2^n classes, where n<3. We discuss herein the operations of the routers **16** to assign packets to the various classes. Further, we discuss the operations that the routers **16** and/or the routers **17** perform to determine whether to retain or discard a packet and/or when to transmit a packet.

Referring now to FIG. **2**, a router **16** includes a classifier **18** that associates a received packet with one of 2^{n+m} classes of service, based primarily on the COS tag, if any, included in the packet header. The classifier maps the 2^{n+m} classes of service to the 2^n classes based on, for example, the highest order n bits or the lowest order n bits of the COS tag. The classifier then uses the remaining bits of the COS tag to set the loss priorities of the packets. As discussed below, the loss priorities determine if respective packets are discarded or retained during times of congestion. The higher the loss priority of a packet, the less likely the packet will be retained.

If the endstation **12** or node **14** that introduces the packet to the network does not support the 802.1p standard, the COS tag may not be included in the packet. The classifier **18** may then assign the packet to one of the 2^{n+m} classes, currently 2^3 classes, based on appropriate network or service provider transmission rules. It may, for example, assign the packet to a "best effort" class. Alternatively, the router **16** may assign the packet to a particular class of service based on a media access control, or MAC, address included in the packet. The classifier then writes the appropriate COS tag to the packet header.

Referring now also to FIG. **3**, the router **16** in this exemplary embodiment supports four classes of service, i.e., 2^2 classes. The classifier **18** maps each of the 2^3 classes of service to an appropriate one of the 2^2 classes of service based on the two highest order bits of the 3-bit COS tag. The third, or lowest order bit, is then used to assign a loss priority to the packet. The classifier **18** thus associates a packet that is assigned to class of service **010** with class of service **01** and sets the loss priority of the packet to 0. Further, the classifier **18** associates a packet that is assigned to class of service **011** with class **01** and sets the loss priority of this packet to 1.

Referring further to FIG. **4** once the classifier **18** associates the packets with the various 2^n classes of service and sets the loss priorities (steps **400–402**), a policer **20** enforces network or service provider usage parameter controls by marking, discarding or passing the packets (step **404**). The usage parameter controls are set by a network manager or service provider based on, for example, levels of service purchased by or associated with a user. The user may, for example, purchase a level of service based on the transmission of a maximum number of packets per hour. If the number of packets being sent by the user exceeds this limit, the policer then marks, discards or passes the excess packets depending on the traffic management rules.

If the policer **20** marks an offending packet, it assigns the packet to a higher loss priority within the associated class of service. This increases the likelihood that the packet will be discarded if the network becomes congested. In the example, the policer sets the lowest order bit of the COS tag to 1. If the packet is already assigned the highest loss priority within the class of service, the policer **20** either passes or discards the packet, depending on the traffic management rules. If the packet is passed, the policer **20** may charge the user for the use of excess bandwidth.

As discussed above, the policer **20** operates in accordance with traffic management rules established by a network manager or service provider. In the embodiment described herein the policer determines if a packet exceeds an established limit by using a "jumping window policing scheme." The policer thus sets a police rate of B/T for a user, where B is a burst size and T is a time interval and both B and T are set by the network manager or service provider. The policer then counts the number of octets received from the user over intervals of length T. If the count exceeds B, the arriving packet is marked, passed or dropped, depending on the enforcement mode utilized by the policer. Various limits may be set, such as, for example, limits that vary based on the number of times the associated policing rate is exceeded by a given user and/or based on the various classes of service.

A WRED processor **22** determines which of the remaining packets, i.e., the packets that the policer has not discarded, are to be retained in a buffer **24** that holds the packets for every output port **30** (steps **406–416**). The use of a single buffer is in contrast to prior known routers that use a separate buffer for each output port.

The WRED processor **22** utilizes a modified weighted-random early detection (WRED) scheme. The WRED processor associates with each of the 2^{n+m} classes of service, "C_i," two thresholds, namely, a maximum threshold MAX_{C_i} and a minimum threshold MIN_{C_i}. As discussed below, the thresholds are used by the processor **22** to determine whether to retain or discard a given packet.

The WRED processor **22** keeps track of an average "free queue" depth, which is an average number of available storage locations in the buffer **24**. When the buffer is empty, all of the buffer storage locations are linked to the free queue. As packets are retained, buffer locations, which are generally referred to in 512 byte pages, are removed from the free queue and linked to appropriate class of service per output port queues. When the packets are later transmitted, the buffer locations are removed from the class of service per output port queues and again linked to the free queue.

Each time a packet is received, the WRED processor **22** determines a new weighted average free queue depth A_{NEW}:

$$A_{NEW}=A_{CURRENT}+W(I-A_{CURRENT})$$

where I is the instantaneous size of the free queue, W is the weighting factor and $A_{CURRENT}$ is the current weighted

average free queue depth (step **406**). The weighting factor W is preferably selected such that multiplication is accomplished by shifting the difference value ($1-A_{CURRENT}$). The value $A_{CURRENT}$ is updated at regular intervals with the value of A_{NEW}, such as after every 64B frame time, which approximates the average packet arrival time.

The WRED processor compares the weighted average A_{NEW} with the MAX_{C_i} and MIN_{C_i} values associated with the appropriate one of the 2^{n+m} classes of service. If the weighted average exceeds the MAX_{C_i} value, the WRED processor **22** retains the packet (step **408**). If the weighted average falls below the MIN_{C_i} value, the WRED processor **22** discards the packet (step **410**). If, however, the average falls between MAX_{C_i} and MIN_{C_i} values, the WRED processor calculates a probability of discard, PD:

$$P_D=b_{c_i}-(m_{c_i}*A_{NEW})$$

where b_{c_i} and m_{c_i} are the intercept and slope values associated with the appropriate one of the 2^{n+m} classes of service (step **412**). As shown in FIG. **5**, the probability of discard changes linearly with changes in the weighted average queue depth. A given packet is discarded when the probability of discard P_D exceeds a "random" number that is produced by a pseudo random generator **25** (steps **414–416**). When the weighted average is relatively low, the probability of discard is larger, and thus, the packet is more likely to be discarded.

The slope and intercept values m_{c_i} and b_{c_i} are selected based on trade-offs between keeping links through the router **16** busy and reserving space in the buffer **24** to handle bursts. For higher classes of service the slope and intercept values are selected to be relatively low—such that the probability of discard is low over the entire range from MAX_{c_i} to MIN_{c_i}. The slope and intercept values for the lower classes of service are typically larger, reflecting the greater associated probability of packet loss for the class and the reservation of spaces in the buffer for bursts of packets assigned to the higher classes. The various threshold values, and slope and intercept values are selected such that packet order and probabilities of packet loss are maintained across the 2^{n+m} classes of service.

In prior known routers, implementing a WRED scheme required maintaining average queue depths for all of the classes of service queues at each of the output ports. Thus, for a router to support 8 classes of service over "y" output ports, it had to calculate average queue depths for 8*y separate queues. In the current router **16**, the WRED processor calculates the average depth of a single free queue, regardless of the number of classes of service.

A scheduler **26** implements a 2^n class-based weighted round robin (WRR) scheduling scheme for each output port (step **418**). The scheduler associates an appropriate weighting factor W_{Q_j} with each class of service per output port queue. The scheduler de-queues W_{Q_j} packets for transfer from the Q_j queue associated with one of the 2^n classes of service, and then de-queues $W_{Q_{j+1}}$ packets from the Q_{j+1} queue for subsequent transfer. If the Q_{j+1} queue is empty, the scheduler de-queues an appropriate number of packets from the Q_{j+2} queue, and so forth. The scheduler **26** thus ensures that each one of the 2^n classes of service is associated with an appropriate maximum delay limit and through-put allocation.

The class of service mapping, modified WRED scheme and WRR scheme in combination ensure that packets are transferred through the router **16** as if the router supported the 2^{n+m} classes of service. The router **16**, however, requires less processing and storage overhead than the prior known routers that support the same number of classes, since the router **16** actually supports 2^n classes of service, and uses a single output buffer to do so.

What is claimed is:

1. A router for use in routing packets over a network, the router supporting a plurality, X, of classes of service and including:

A. a plurality of input ports for receiving packets over the network;

B. a plurality of output ports for transferring packets over the network;

C. a classifier for assigning packets received by the input ports to X*Y classes of service, where * represents multiplication, and mapping the XY classes of service to the X classes of service that are supported by the router, the classifier assigning to the packet one of Y associated levels of priority, wherein each level of priority is associated with a different probability of packet loss;

D. means for retaining the packets based on probabilities of discard associated with the X*Y classes of service; and

E. scheduling means for transferring the packets through each of the output ports based on the X classes of service.

2. The router of claim **1** further including a multiple storage location buffer for retaining packets to be transferred through the output ports, the buffer linking the storage locations that contain packets in class of service per output port queues and linking available storage locations in a free queue.

3. The router of claim **2** wherein the means for retaining the packets further includes:

i. means for determining a new weighted average depth for the free queue, and

ii. means for determining a probability of discard for a given packet if the new weighted average queue depth falls below a predetermined maximum threshold associated with the class of service to which the packet is assigned.

4. The router of claim **3** wherein the means for retaining the packets discards a given packet if the associated weighted average depth for the free queue falls below a minimum threshold associated with the class of service to which the packet is assigned.

5. The router of claim **4** wherein the means for retaining the packets calculates the probability of discard as $P_d=c-(m*A_{NEW})$ where c is an intercept and m is a slope that is associated with a line that plots a weighted average free queue depth versus probability of discard for the class of service to which the packet is assigned, and A_{NEW} is the weighted average depth of the free queue.

6. The router of claim **5** wherein the means for retaining the packets calculates the new weighted average depth of the free queue as $A_{NEW}=A_{CURRENT}+w(I-A_{CURRENT})$ where w is a weighting factor, I represents the instantaneous depth of the free queue and $A_{CURRENT}$ is the current weighted average depth of the free queue.

7. The router of claim **6** wherein the scheduling means selects packets for transfer based on weighting factors associated with the respective X classes of service.

8. The router of claim **1** wherein the router supports $X=2^n$ classes of service and the classifier assigns packets to $X*Y=2^{n+m}$ classes of service.

9. A router for use in routing packets over a network, the router supporting a plurality, X, of classes of service and including:

A. a plurality of input ports for receiving packets over the network;

B. a plurality of output ports for transferring packets over the network;

C. a multiple storage location buffer for retaining packets to be transferred through the output ports;

D. means for retaining the packets based on probabilities of discard associated with X*Y classes of service where * represents multiplication; and

E. scheduling means for transferring the packets through each of the output ports based on the X classes of service that the router supports.

10. The router of claim **9** further including a classifier for:

i. assigning packets received by the input ports to X*Y classes of service,

ii. associating the packets with the X classes of service that are supported by the router, and

iii. assigning to the packet one of Y associated levels of priority, wherein each level of priority is associated with a different probability of packet loss.

11. The router of claim **10** wherein the means for retaining the packets further includes

i. means for determining a new weighted average queue depth for a free queue that links available buffer storage locations, and

ii. means for determining a probability of discard for a given packet if the new weighted average free queue depth falls below a predetermined maximum threshold associated with the class of service to which the packet is assigned.

12. the router of claim **11** wherein the means for retaining the packets calculates the probability of discard as $P_d=c-(m*A_{NEW})$ where c is an intercept and m is a slope that are associated with a line that plots a weighted average free queue depth versus probability of discard for the class of service to which the packet is assigned, and A_{NEW} is the weighted average depth of the free queue.

13. The router of claim **12** wherein the means for retaining the packets calculates the weighted average depth of the free queue as $A_{NEW}=A_{CURRENT}+w(I-A_{CURRENT})$ where w is a weighted factor, I represents the instantaneous depth of the free queue and $A_{CURRENT}$) is the current weighted average depth of the free queue.

14. The router of claim **13** wherein the means for retaining the packets discards a given packet if the new weighted average free queue depth falls below a minimum threshold associated with the class of service to which the packet is assigned.

15. The router of claim **13** wherein the means for retaining the packets retains a given packet if the new weighted average free queue depth is above a maximum threshold associated with the class of service to which the packet is assigned.

16. The router of claim **9** wherein the scheduling means selects packets for transfer through each output port based on weighting factors associated with the respective X classes of service.

17. The router of claim **16** wherein the buffer links retained packets in class of service per output port queues and the scheduling means selects packets from the class of service per output port queues.

18. The router of claim **9** wherein the router supports $X=2^n$ classes of service and the means for retaining retains packets bases on probabilities of discard associated with $X*Y=2^{n+m}$ classes of service.

19. A method of routing packets through a router that supports a plurality, X, of classes of service, the method including the steps of:

A. receiving packets through one or more input ports;

B. assigning packets received by the input ports to X*Y classes of service, where * represents multiplication, mapping the X*Y classes of service to the X classes of service that are supported by the router, and assigning to the packet one of Y associated levels of priority, wherein each level of priority is associated with a different probability of packet loss;

C. retaining the packets based on probabilities of discard associated with the X*Y classes of service; and

D. transferring the packets through one or more output ports based on the X classes of service.

20.The method of routing packets of claim **19** further including in the step of retaining the packets the steps of:

i. retaining the packets in a multiple storage location buffer and linking available storage locations to a free queue,

ii. determining a new weighted average depth for the free queue, and

iii. determining a probability of discard for a given packet if the new weighted average queue depth falls below a predetermined maximum threshold associated with the class of service to which the packet is assigned.

21. The method of routing packets of claim **20** including in the step of retaining the packets the further step of discarding a given packet if the new weighted average depth for the free queue falls below a minimum threshold associated with the class of service to which the packet is assigned.

22. The method of routing packets of claim **21** wherein the step of retaining the packets includes calculating the probability of discard as $P_d=c-(m*A_{NEW})$ where c is an intercept and m is a slope associated with a line that plots weighted average free queue depth versus probability of discard for the class of service to which the packet is assigned, and A_{NEW} is the new weighted average depth of the free queue.

23. The method of routing packets of claim **22** wherein the step of retaining the packets includes calculating the new weighted average depth of the free queue as $A_{NEW}=A_{CURRENT}+w(I-A_{CURRENT})$ where w is a weighting factor I represents the instantaneous depth of the free queue and $A_{CURRENT}$ is the current weighted average queue depth.

24. The method of claim **23** wherein the discard means retains a given packet of the new weighted average free queue depth is above a maximum threshold associated with the class of service to which the packet is assigned.

25. The method of routing packets of claim **19** wherein the step of transferring packets through the more or more output port transfers the packets based on weighting factors associated with the respective X classes of service.

26. The method of routing packets of claim **19** wherein the router supports $X=2^n$ classes of service and, in the step of assigning packets, the packets are assigned to $X*Y=2^{n+m}$ classes of service.

27. A method of routing packets through a router that supports a plurality, X, of classes of service, the method including:

A. receiving packets through one or more input ports and assigning the packets to X*Y classes of service, where * represents multiplication;

B. retaining packets based on probabilities of discard associated with the X*Y classes of service in a multiple storage location buffer that links available storage locations to a free queue; and

C. transferring the packets through one or more output ports based on the X classes of service.

28. The method of routing of claim **27** further including the steps:

i. associating the packets that are assigned to the X*Y classes of service with the X classes of service that are supported by the router, and

ii. assigning to the respective packets one of Y associated levels of priority, wherein each level of priority associated with a different probability of packet loss.

29. The method of routing packets of claim **28** wherein the step of retaining the packets includes:

a. determining a new weighted average depth for the free queue, and

b. determining a probability of discard for a given packet if the new weighted average free queue depth falls below a predetermined maximum threshold associated with the class of service to which the packet is assigned.

30. The method of routing packets of claim **29** wherein the step of retaining packets further includes calculating the probability of discard as $P_d=c-(m*A_{NEW})$ where c is an intercept and m is a slope that are associated with a line that plots average free queue depth versus probability of discard for the class of service to which the packet is assigned, and A_{NEW} is the new weighted average depth of the free queue.

31. The method of routing packets of claim **30** wherein the step of retaining packets further includes calculating the new weighted average depth of the free queue as $A_{NEW}=A_{CURRENT}+w(I-A_{CURRENT})$ where w is a weighting factor, I represents the instantaneous depth of the free queue and $A_{CURRENT}$ is the current weighted average queue depth.

32. The method of routing packets of claim **29** wherein the step of retaining packets further includes discarding a given packet if the new weighted average free queue depth falls below a minimum threshold associated with the class of service to which the packet is assigned.

33. The method of routing packets of claim **29** wherein the step of retaining packets further includes retaining a given packet if the new weighted average free queue depth is above the maximum threshold associated with the class of service to which the packet is assigned.

34. The method of routing packets of claim **27** wherein the router supports $X=2^n$ classes of service and, in the step of retaining packets, the packets are retained based on probabilities of discard associated with $X*Y=2^{n+m}$ classes of service.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

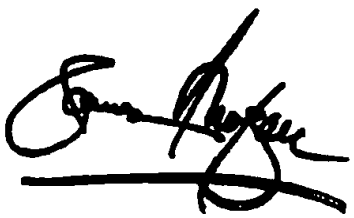
PATENT NO. : 6,463,068 B1
DATED : October 8, 2002
INVENTOR(S) : Lin et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,
Item [73], should read:
-- Assignee: **Cisco Technology, Inc.**, San Jose, CA (US) --

Signed and Sealed this

Twenty-first Day of January, 2003



JAMES E. ROGAN
Director of the United States Patent and Trademark Office

US006591299B2

(12) **United States Patent**
Riddle et al.

(10) **Patent No.: US 6,591,299 B2**
(45) **Date of Patent: *Jul. 8, 2003**

(54) **METHOD FOR AUTOMATICALLY CLASSIFYING TRAFFIC WITH ENHANCED HIERARCHY IN A PACKET COMMUNICATIONS NETWORK**

(75) Inventors: **Guy Riddle**, Los Gatos, CA (US); **Robert L. Packer**, Rancho Santa Fe, CA (US); **Mark Hill**, Los Altos, CA (US)

(73) Assignee: **Packeteer, Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/155,936**

(22) Filed: **May 24, 2002**

(65) **Prior Publication Data**

US 2002/0143939 A1 Oct. 3, 2002

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/990,354, filed on Nov. 23, 2001, now Pat. No. 6,457,051, which is a continuation of application No. 09/198,090, filed on Nov. 23, 1998, now Pat. No. 6,412,000.

(60) Provisional application No. 60/066,864, filed on Nov. 25, 1997.

(51) **Int. Cl.**[7] **G06F 15/173**

(52) **U.S. Cl.** **709/224**; 709/223; 709/230; 709/234; 709/238; 709/242; 370/230; 370/235; 370/252; 370/351; 370/355; 370/356

(58) **Field of Search** 709/200, 220–226, 709/230, 232, 234–236, 238–239, 242, 245–246; 370/224–232, 235–237, 252–255, 351–356

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,251,152	A		10/1993	Notess
5,495,426	A		2/1996	Waclawsky et al.
5,539,659	A	*	7/1996	McKee et al. 709/224
5,781,726	A	*	7/1998	Pereira 709/200
5,838,919	A		11/1998	Schwaller et al.
5,870,561	A		2/1999	Jarvis et al.
5,903,559	A		5/1999	Acharya et al.
5,923,849	A		7/1999	Venkatraman
6,028,842	A		2/2000	Chapman et al.
6,046,980	A		4/2000	Packer
6,092,115	A	*	7/2000	Choudhury et al. 709/235
6,137,782	A		10/2000	Sharon et al.
6,205,121	B1	*	3/2001	Heuer 370/250
6,209,033	B1		3/2001	Datta et al.
6,262,976	B1	*	7/2001	McNamara 709/220
6,263,368	B1	*	7/2001	Martin 709/224
6,412,000	B1	*	6/2002	Riddle et al. 709/224
6,457,051	B1	*	9/2002	Riddle et al. 709/224

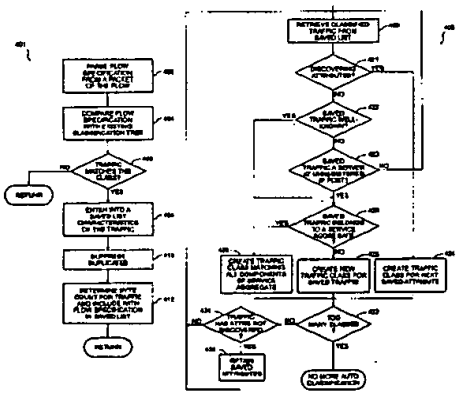
* cited by examiner

Primary Examiner—Bharat Barot
(74) *Attorney, Agent, or Firm*—Townsend and Townsend and Crew LLP; Kenneth R. Allen

(57) **ABSTRACT**

In packet communication, a method for automatically classifying packet flows for use in allocating bandwidth resources and the like by a rule of assignment of a service level. By rendering discoverable the attributes of a flow specification for packet flows, a finer grained hierarchy of classification is provided automatically that is based on information which is specific to the type of program or application supported by the flow and thus allowing greater flexibility in control over different flows within the same application. The method comprises applying individual instances of traffic classification paradigms to packet network flows based on selectable information obtained from a plurality of layers to define a characteristic class, then mapping the flow to the defined traffic class. The flow specification is provided with some application-specific attributes, some of which are discoverable. The discoverable attributes lead to an ability to automatically create sub-nodes of nodes for finer-grained control.

17 Claims, 7 Drawing Sheets



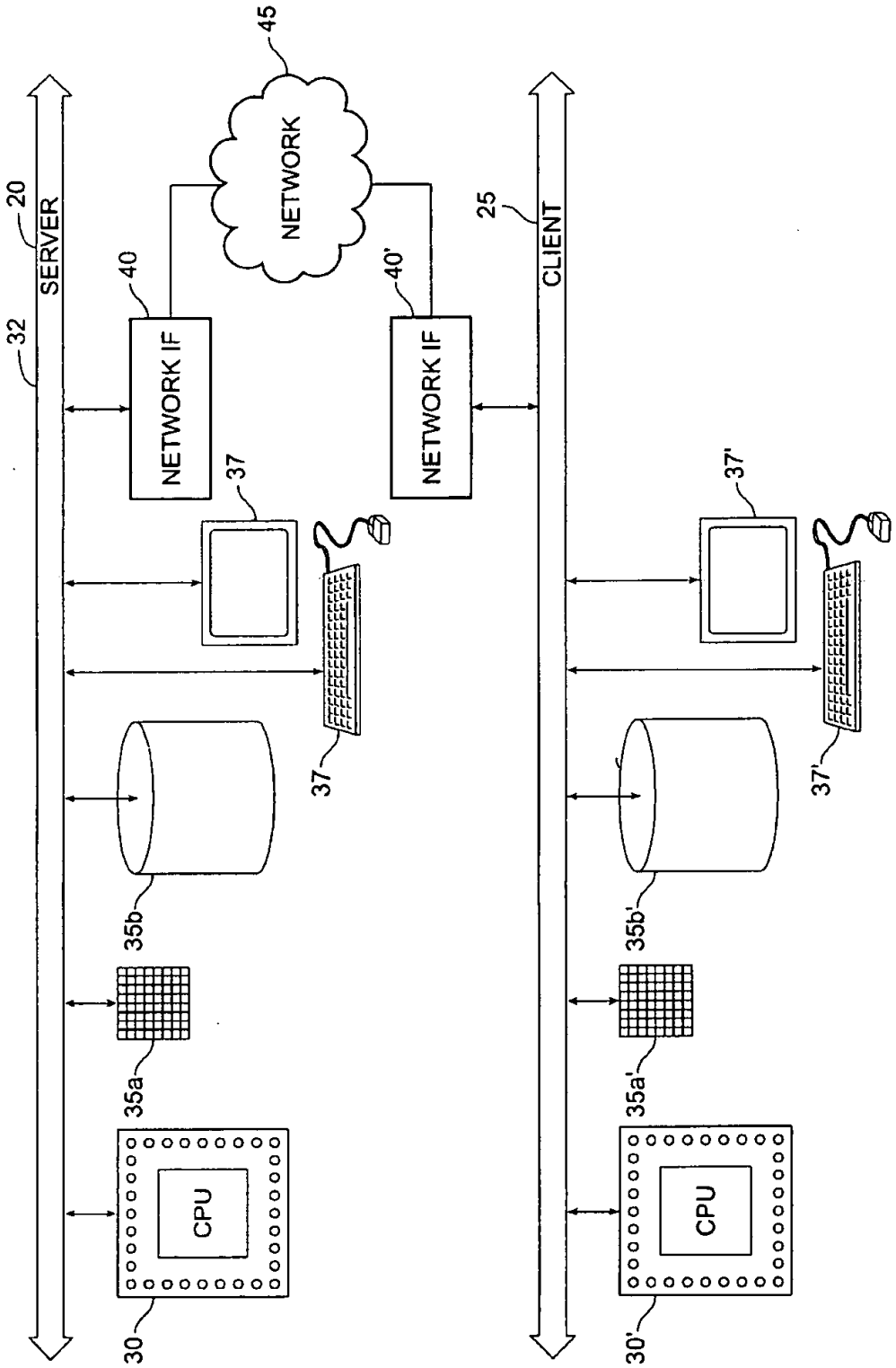


FIG. 1A
(PRIOR ART)

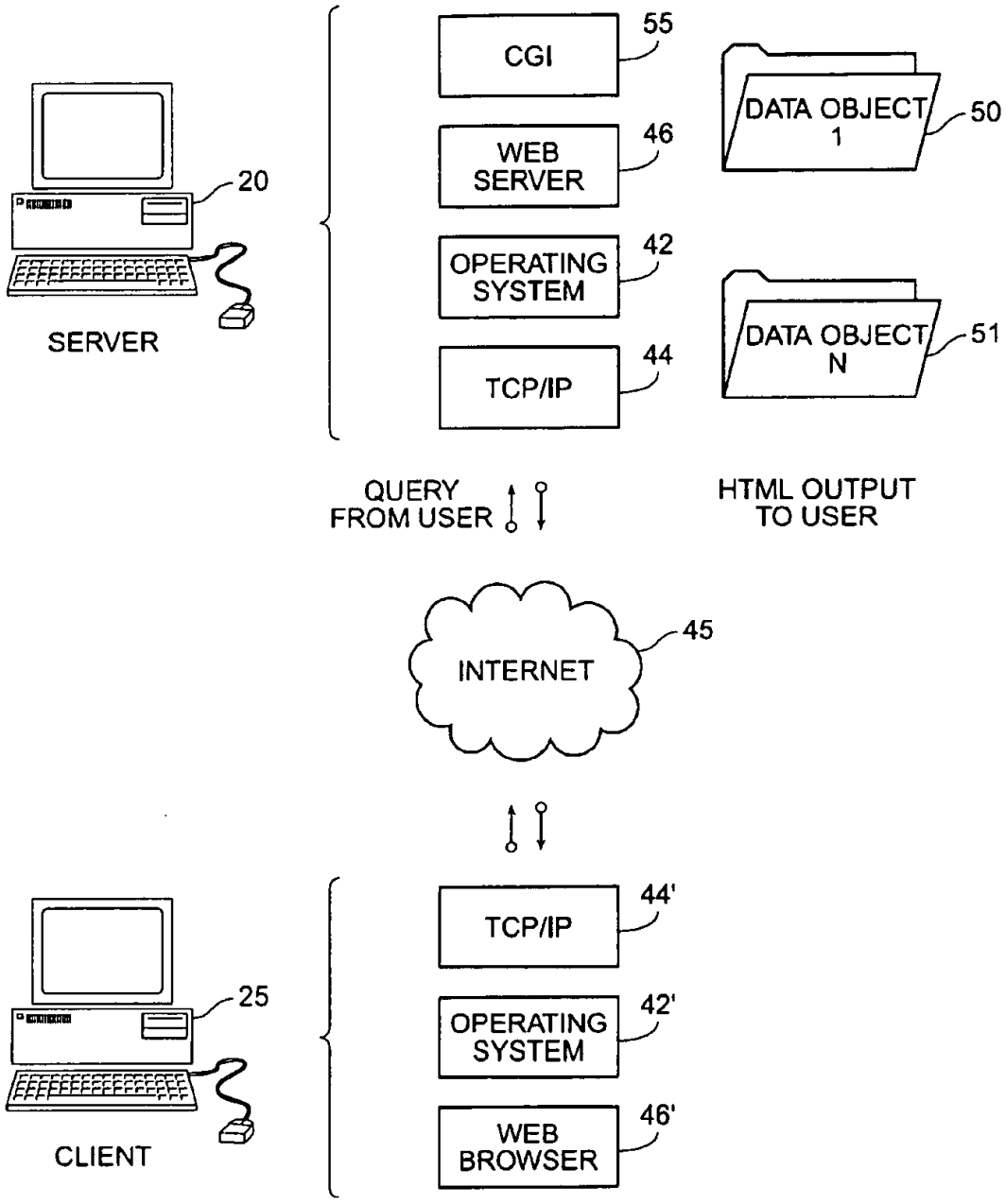


FIG. 1B
(PRIOR ART)

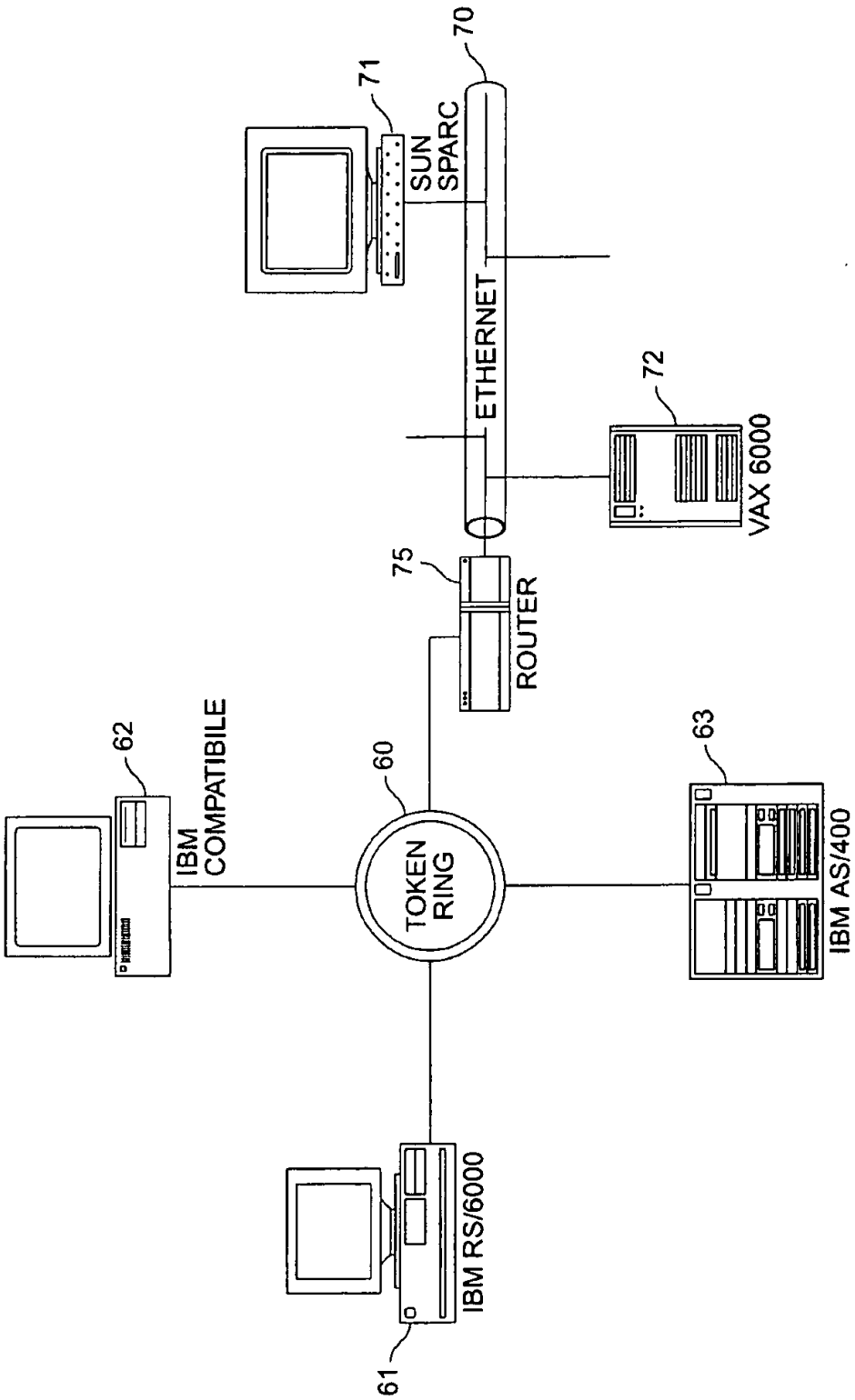
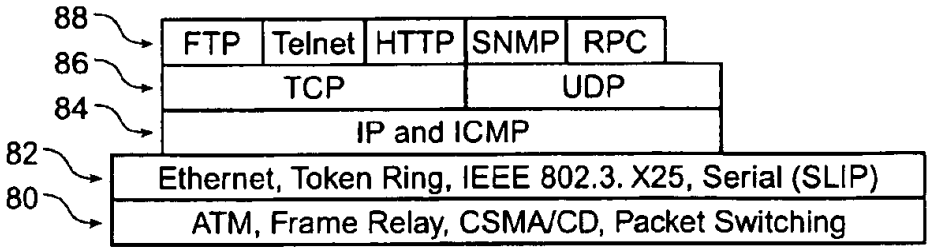


FIG. 1C
(PRIOR ART)



LEGEND

88 Session/Application Layer
86 Transport Layer
84 Network Layer
82 Data Link Layer
80 Physical Layer

FIG. 1D
(PRIOR ART)

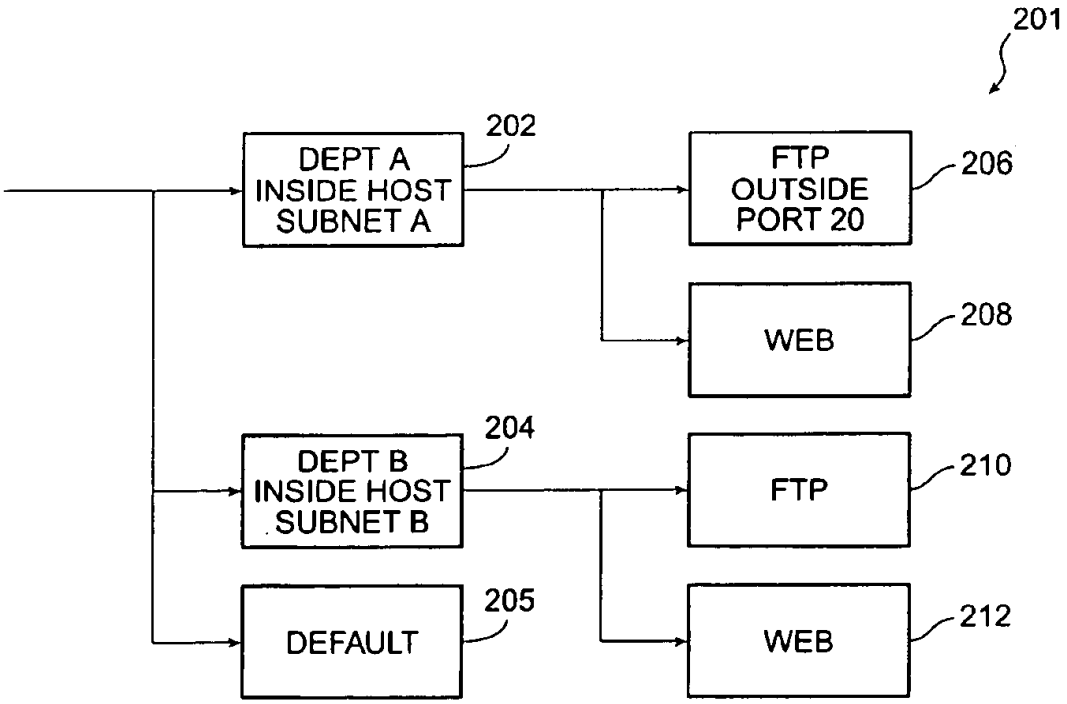


FIG. 2A

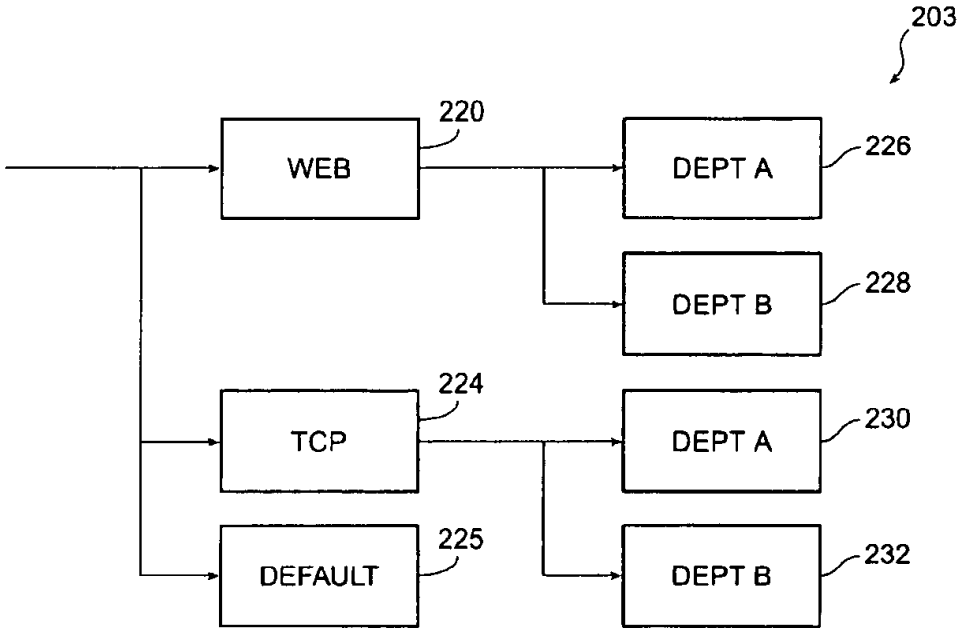

203
220
WEB
DEPT A
226
DEPT B
228
224
TCP
DEPT A
230
DEFAULT
225
DEPT B
232

FIG. 2B

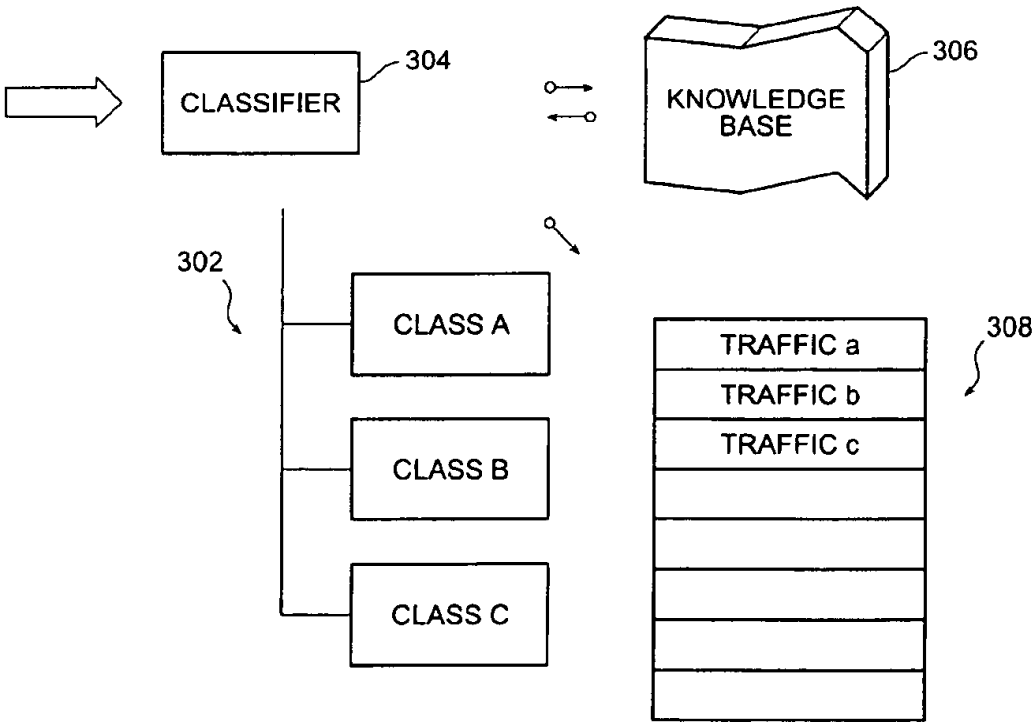

CLASSIFIER
304
KNOWLEDGE BASE
306
302
CLASS A
CLASS B
CLASS C
TRAFFIC a
TRAFFIC b
TRAFFIC c
308

FIG. 3

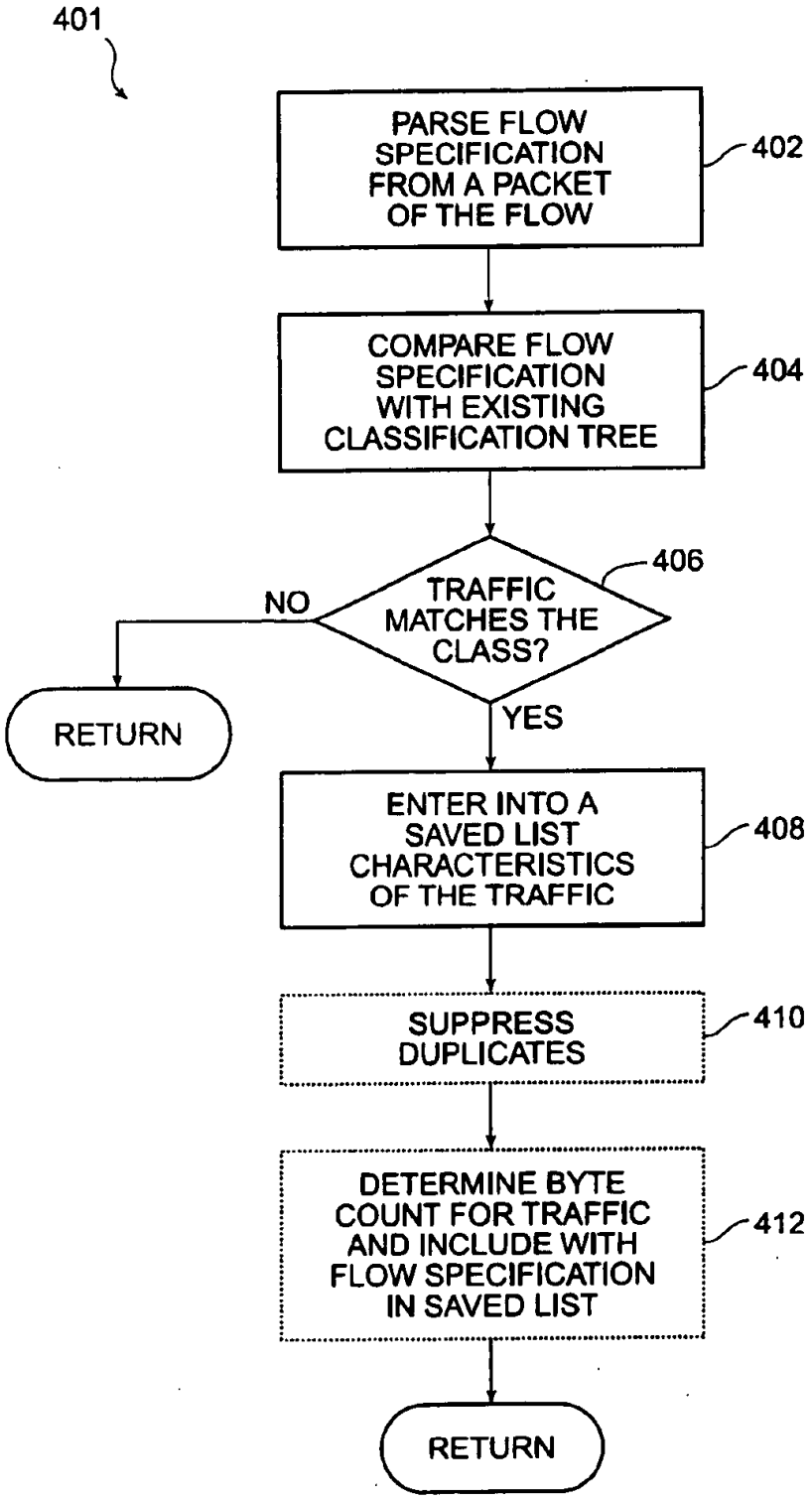

401
PARSE FLOW SPECIFICATION FROM A PACKET OF THE FLOW
402
COMPARE FLOW SPECIFICATION WITH EXISTING CLASSIFICATION TREE
404
TRAFFIC MATCHES THE CLASS?
406
NO
YES
RETURN
ENTER INTO A SAVED LIST CHARACTERISTICS OF THE TRAFFIC
408
SUPPRESS DUPLICATES
410
DETERMINE BYTE COUNT FOR TRAFFIC AND INCLUDE WITH FLOW SPECIFICATION IN SAVED LIST
412
RETURN

FIG. 4A

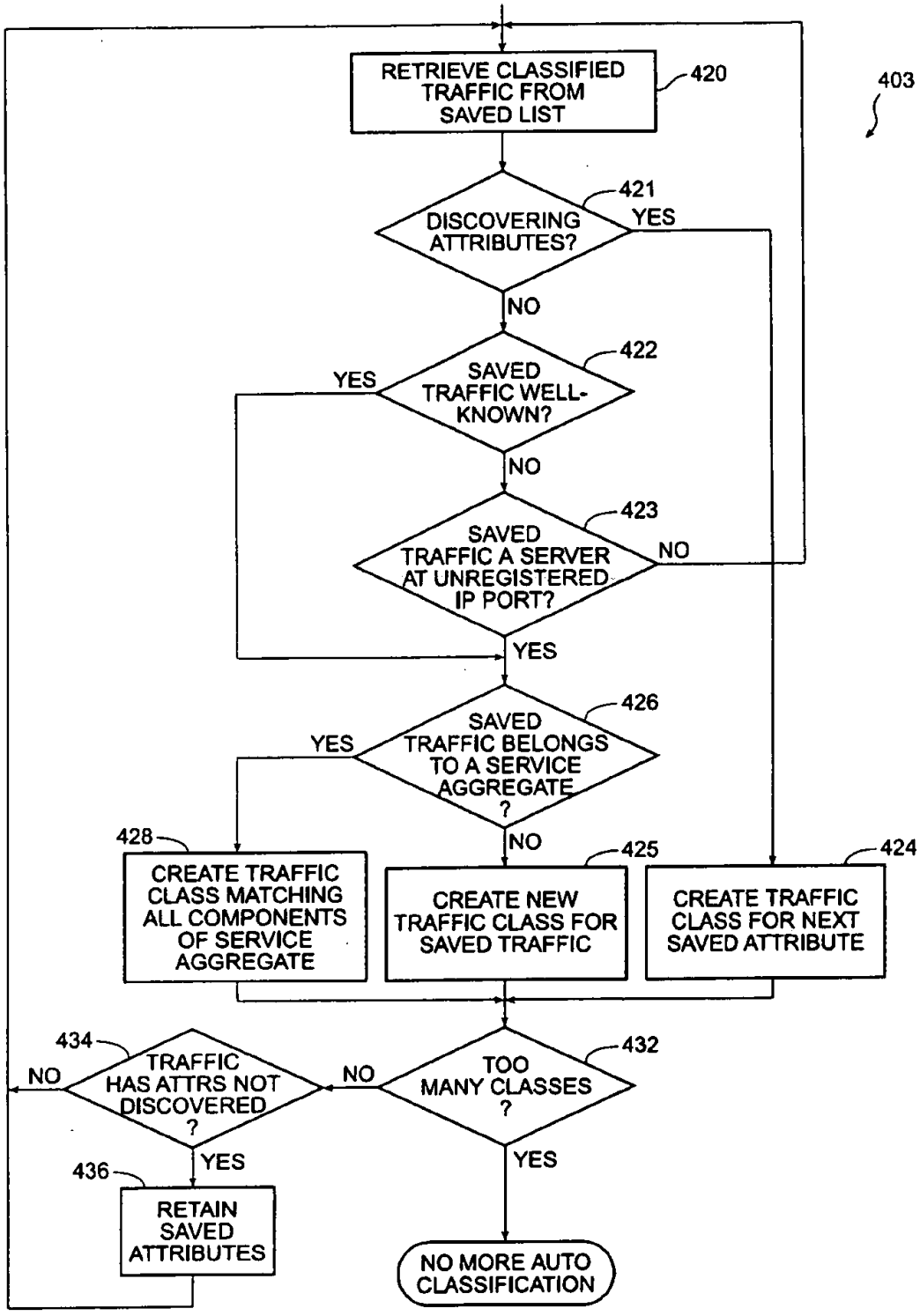

RETRIEVE CLASSIFIED TRAFFIC FROM SAVED LIST
420
403
DISCOVERING ATTRIBUTES?
421
YES
NO
SAVED TRAFFIC WELL-KNOWN?
422
YES
NO
SAVED TRAFFIC A SERVER AT UNREGISTERED IP PORT?
423
NO
YES
SAVED TRAFFIC BELONGS TO A SERVICE AGGREGATE ?
426
YES
NO
428
CREATE TRAFFIC CLASS MATCHING ALL COMPONENTS OF SERVICE AGGREGATE
425
CREATE NEW TRAFFIC CLASS FOR SAVED TRAFFIC
424
CREATE TRAFFIC CLASS FOR NEXT SAVED ATTRIBUTE
434
TRAFFIC HAS ATTRS NOT DISCOVERED ?
NO
TOO MANY CLASSES ?
432
NO
YES
YES
436
RETAIN SAVED ATTRIBUTES
NO MORE AUTO CLASSIFICATION

FIG. 4B

METHOD FOR AUTOMATICALLY CLASSIFYING TRAFFIC WITH ENHANCED HIERARCHY IN A PACKET COMMUNICATIONS NETWORK

CROSS-REFERENCES TO RELATED APPLICATIONS

This application claims priority from a commonly owned U.S. Provisional Patent Application, Serial No. 60/066,864, filed Nov. 25, 1997, in the name of Guy Riddle and Robert L. Packer, entitled "Method for Automatically Classifying Traffic in a Policy Based Bandwidth Allocation System."

This is a continuation-in-part of U.S. application Ser. No. 09/990,354 filed Nov. 23, 2001, now U.S. Pat. No. 6,457,051, in the name of Guy Riddle and Robert L. Packer, entitled Method For Automatically Classifying Traffic In A Packet Communications Network, which is a continuation of application Ser. No. 09/198,090 filed Nov. 23, 1998, now U.S. Pat. No. 6,412,000, also in the name of Guy Riddle and Robert L. Packer, also entitled Method For Automatically Classifying Traffic In A Packet Communications Network.

The following related commonly-owned U.S. patent application is hereby incorporated by reference in its entirety for all purposes: U.S. patent application Ser. No. 09/198,051, filed Nov. 23, 1998, still pending, in the name of Guy Riddle, entitled "Method for Automatically Determining a Traffic Policy in a Packet Communications Network."

Further, this application makes reference to the following commonly owned U.S. Patents and Applications, which are incorporated by reference herein in their entirety for all purposes:

U.S. Pat. No. 5,802,106, in the name of Robert L. Packer, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision," relates to a technique for automatically determining the data rate of a TCP connection;

U.S. patent application Ser. No. 08/742,994, now U.S. Pat. No. 6,038,216, in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Environment Without a Data Rate Supervision," relates to a technique for automatically scheduling TCP packets for transmission;

U.S. Pat. No. 6,046,980, in the name of Robert L. Packer, entitled "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers in Store and Forward Network," relates to a technique for automatically allocating bandwidth based upon data rates of TCP connections according to a hierarchical classification paradigm; and

U.S. patent application Ser. No. 08/742,994 now U.S. Pat. No. 6,038,216 issued Mar. 14, 2000, in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Environment Without a Data Rate Supervision," relates to a technique for automatically scheduling TCP packets for transmission.

STATEMENT AS TO RIGHTS TO INVENTIONS MADE UNDER FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

NOT APPLICABLE

REFERENCE TO A "SEQUENCE LISTING," A TABLE, OR A COMPUTER PROGRAM LISTING APPENDIX SUBMITTED ON A COMPACT DISK

NOT APPLICABLE

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

This invention relates to digital packet telecommunications, and particularly to management of network bandwidth based on information ascertainable from multiple layers of OSI network model. It is particularly useful in conjunction with bandwidth allocation mechanisms employing traffic classification in a digitally-switched packet telecommunications environment, as well as in monitoring, security and routing.

The ubiquitous TCP/IP protocol suite, which implements the world-wide data communication network environment called the Internet and is also used in private networks (Intranets), intentionally omits explicit supervisory function over the rate of data transport over the various media which comprise the network. While there are certain perceived advantages, this characteristic has the consequence of juxtaposing very high-speed packet flows and very low-speed packet flows in potential conflict for network resources, which results in inefficiencies. Certain pathological loading conditions can result in instability, overloading and data transfer stoppage. Therefore, it is desirable to provide some mechanism to optimize efficiency of data transfer while minimizing the risk of data loss. Early indication of the rate of data flow which can or must be supported is imperative. In fact, data flow rate capacity information is a key factor for use in resource allocation decisions. For example, if a particular path is inadequate to accommodate a high rate of data flow, an alternative route can be sought out.

Internet/Intranet technology is based largely on the TCP/IP protocol suite, where IP, or Internet Protocol, is the network layer protocol and TCP, or Transmission Control Protocol, is the transport layer protocol. At the network level, IP provides a "datagram" delivery service. By contrast, TCP builds a transport level service over the datagram service to provide guaranteed, sequential delivery of a byte stream between two IP hosts.

TCP flow control mechanisms operate exclusively at the end stations to limit the rate at which TCP endpoints emit data. However, TCP lacks explicit data rate control. The basic flow control mechanism is a sliding window, superimposed on a range of bytes beyond the last explicitly-acknowledged byte. Its sliding operation limits the amount of unacknowledged transmissible data that a TCP endpoint can emit.

Another flow control mechanism is a congestion window, which is a refinement of the sliding window scheme, which employs conservative expansion to fully utilize all of the allowable window. A component of this mechanism is sometimes referred to as "slow start".

The sliding window flow control mechanism works in conjunction with the Retransmit Timeout Mechanism (RTO), which is a timeout to prompt a retransmission of unacknowledged data. The timeout length is based on a running average of the Round Trip Time (RTT) for acknowledgment receipt, i.e. if an acknowledgment is not received within (typically) the smoothed RTT+4·mean deviation, then packet loss is inferred and the data pending acknowledgment is retransmitted.

Data rate flow control mechanisms which are operative end-to-end without explicit data rate control draw a strong

inference of congestion from packet loss (inferred, typically, by RTO). TCP end systems, for example, will "back-off", i.e., inhibit transmission in increasing multiples of the base RTT average as a reaction to consecutive packet loss.

Bandwidth Management in TCP/IP Networks

Conventional bandwidth management in TCP/IP networks is accomplished by a combination of TCP end systems and routers which queue packets and discard packets when certain congestion thresholds are exceeded. The discarded, and therefore unacknowledged, packet serves as a feedback mechanism to the TCP transmitter. (TCP end systems are clients or servers running the TCP transport protocol, typically as part of their operating system.)

The term "bandwidth management" is often used to refer to link level bandwidth management, e.g. multiple line support for Point to Point Protocol (PPP). Link level bandwidth management is essentially the process of keeping track of all traffic and deciding whether an additional dial line or ISDN channel should be opened or an extraneous one closed. The field of this invention is concerned with network level bandwidth management, i.e. policies to assign available bandwidth from a single logical link to network flows.

In U.S. Pat. No. 6,038,216, in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Environment Without Data Rate Supervision," a technique for automatically scheduling TCP packets for transmission is disclosed. Furthermore, in U.S. Pat. No. 5,802,106, in the name of Robert L. Packer, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision," a technique for automatically determining the data rate of a TCP connection is disclosed. Finally, in a U.S. patent application Ser. No. 08/977,376, now U.S. Pat. No. 6,046,980, in the name of Robert L. Packer, entitled "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers in Store and Forward Network," a technique for automatically allocating bandwidth based upon data rates of TCP connections according to a hierarchical classification paradigm is disclosed.

Automated tools assist the network manager in configuring and managing the network equipped with the rate control techniques described in these copending applications. In a related copending application, a tool is described which enables a network manager to automatically produce policies for traffic being automatically detected in a network. It is described in a copending U.S. patent application Ser. No. 09/198,051, still pending in the name of Guy Riddle, entitled "Method for Automatically Determining a Traffic Policy in a Packet Communications Network," based on U.S. Provisional Patent Application Serial No. 60/066,864. The subject of the present invention is also a tool designed to assist the network manager.

While these efforts teach methods for solving problems associated with scheduling transmissions, automatically determining data flow rate on a TCP connection, allocating bandwidth based upon a classification of network traffic and automatically determining a policy, respectively, there is no teaching in the prior art of methods for automatically classifying packet traffic based upon information gathered from a multiple layers in a multi-layer protocol network.

Bandwidth has become an expensive commodity as traffic expands faster than resources and the need to "prioritize" a scarce resource becomes ever more critical. One way to solve this is by applying "policies" to control traffic classified as to type of service required in order to more efficiently match resources with traffic.

Traffic may be classified by type, e.g. E-mail, web surfing, file transfer, at various levels. For example, to classify by network paradigm, examining messages for an IEEE source/destination service access point (SAP) or a sub-layer access protocol (SNAP) yields a very broad indicator, i.e., SNA or IP. More specific types exist, such as whether an IP protocol field in an P header indicates TCP or UDP. Well known connection ports provide indications at the application layer, i.e., SMTP or HTTP.

Classification is not new. Firewall products like "Check-Point FireWall-1," a product of CheckPoint Software Technologies, Inc., a company with headquarters in Redwood City, Calif., have rules for matching traffic. Prior bandwidth managers classify by destination. The PacketShaper, a product of Packeteer, Inc., a company with headquarters in Cupertino, Calif., allows a user to manually enter rules to match various traffic types for statistical tracking, i.e., counting by transaction, byte count, rates, etc. However, manual rule entry requires a level of expertise that limits the appeal for such a system to network savvy customers. What is really needed is a method for analyzing real traffic in a customer's network and automatically producing a list of the "found traffic."

SUMMARY OF THE INVENTION

According to the invention, in a packet communication environment, a method is provided for automatically classifying packet flows for use in allocating bandwidth resources and the like by a rule of assignment of a service level. By rendering discoverable the attributes of a flow specification for packet flows, a finer grained hierarchy of classification is provided automatically that is based on information which is specific to the type of program or application supported by the flow and thus allowing greater flexibility in control over different flows within the same application. The method comprises applying individual instances of traffic classification paradigms to packet network flows based on selectable information obtained from a plurality of layers of a multi-layered communication protocol in order to define a characteristic class, then mapping the flow to the defined traffic class. The flow specification is provided with some application-specific attributes, some of which are discoverable. The discoverable attributes lead to an ability to automatically create sub-nodes of nodes for finer-grained control. The automatic classification is sufficiently robust to classify a complete enumeration of the possible traffic.

In the present invention network managers need not know the technical aspects of each kind of traffic in order to configure traffic classes and service aggregates bundle traffic to provide a convenience to the user, by clarifying processing and enables the user to obtain group counts of all parts comprising a service.

The invention will be better understood upon reference to the following detailed description in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A depicts a representative client server relationship in accordance with a particular embodiment of the invention;

FIG. 1B depicts a functional perspective of the representative client server relationship in accordance with a particular embodiment of the invention;

FIG. 1C depicts a representative internetworking environment in accordance with a particular embodiment of the invention;

FIG. 1D depicts a relationship diagram of the layers of the TCP/IP protocol suite;

FIGS. 2A–2B depict representative divisions of bandwidth;

FIG. 3 depicts a component diagram of processes and data structures in accordance with a particular embodiment of the invention; and

FIGS. 4A–4B depict flowcharts of process steps in automatically classifying traffic in accordance with a particular embodiment of the invention.

DESCRIPTION OF SPECIFIC EMBODIMENTS OF THE INVENTION

1.0 Introduction

The present invention provides techniques to automatically classify a plurality of heterogeneous packets in a packet telecommunications system for management of network bandwidth in systems such as a private area network, a wide area network or an internetwork. Systems according to the present invention enable network managers to: automatically define traffic classes, for which policies may then be created for specifying service levels for the traffic classes and isolating bandwidth resources associated with certain traffic classes. Inbound as well as outbound traffic may be managed. Below is a definitional list of terminology used herein.

List of Definitional Terms

ADMISSIONS CONTROL A policy invoked whenever a system according to the invention detects that a guaranteed information rate cannot be maintained. An admissions control policy is analogous to a busy signal in the telephone world.

CLASS SEARCH ORDER A search method based upon traversal of a N-ary tree data structure containing classes.

COMMITTED INFORMATION RATE (CIR) A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for a committed bandwidth. Also called a guaranteed information rate (GIR).

EXCEPTION A class of traffic provided by the user which supersedes an automatically determined classification order.

EXCESS INFORMATION RATE (EIR) A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for uncommitted bandwidth resources.

FLOW A flow is a single instance of a connection or packet-exchange activity. For example, all packets in a TCP connection belong to the same flow, as do all packets in a UDP session. A flow always is associated with a traffic class.

GUARANTEED INFORMATION RATE (GIR) A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for a committed bandwidth. Also called a committed information rate (CIR).

INSIDE On the LAN side of the bandwidth management device.

MATCHING RULE A description which is used to determine whether a flow matches a traffic class, e.g., "inside service:http", which will match any flows which are connected to an HTTP server on the "inside" of the bandwidth management device. Also known as "traffic specifications".

OUTSIDE On the WAN or Internet side of the bandwidth management device.

PARTITION Partition is an arbitrary unit of network resources.

POLICY A rule assigned to a given class that defines how the traffic associated with the class will be handled during bandwidth management.

POLICY INHERITANCE A method for assigning policies to flows for which no policy exists in a hierarchical arrangement of policies. For example, if a flow matches the traffic class for FTP traffic to Host A, and no corresponding policy exists, a policy associated with a less specific node, such as the traffic class which matches FTP traffic to any host, may be located and used.

POLICY BASED SCALING An adjustment of a requested data rate for a particular flow based upon the policy associated with the flow and information about the flow's potential rate.

SCALED RATE Assignment of a data rate based upon detected speed.

SERVICE LEVEL A service paradigm having a combination of characteristics defined by a network manager to handle a particular class of traffic. Service levels may be designated as either reserved or unreserved.

TRAFFIC CLASS A logical grouping of traffic flows that share the same characteristics—such as application, protocol, address, or set of addresses. A traffic class is defined with a series of matching rules.

TRAFFIC SPECIFICATION See "matching rule".

URI A Universal Resource Identifier is the name of the location field in a web reference address. It is also called a URL or Universal Resource Locator

1.1 Hardware Overview

The method for classifying heterogeneous packets in a packet telecommunications environment of the present invention may be implemented in the C programming language and made operational on a computer system such as shown in FIG. 1A. This invention may be implemented in a client-server environment, but a client-server environment is not essential. This figure shows a conventional client-server computer system which includes a server **20** and numerous clients, one of which is shown as client **25**. The use of the term "server" is used in the context of the invention, wherein the server receives queries from (typically remote) clients, does substantially all the processing necessary to formulate responses to the queries, and provides these responses to the clients. However, server **20** may itself act in the capacity of a client when it accesses remote databases located at another node acting as a database server.

The hardware configurations are in general standard and will be described only briefly. In accordance with known practice, server **20** includes one or more processors **30** which communicate with a number of peripheral devices via a bus subsystem **32**. These peripheral devices typically include a storage subsystem **35**, comprised of a memory subsystem **35***a* and a file storage subsystem **35***b* holding computer programs (e.g., code or instructions) and data, a set of user interface input and output devices **37**, and an interface to outside networks, which may employ Ethernet, Token Ring, ATM, IEEE 802.3, ITU X.25, Serial Link Internet Protocol (SLIP) or the public switched telephone network. This interface is shown schematically as a "Network Interface" block **40**. It is coupled to corresponding interface devices in client computers via a network connection **45**.

Client **25** has the same general configuration, although typically with less storage and processing capability. Thus, while the client computer could be a terminal or a low-end personal computer, the server computer is generally a high-end workstation or mainframe, such as a SUN SPARC server. Corresponding elements and subsystems in the client computer are shown with corresponding, but primed, reference numerals.

Bus subsystem **32** is shown schematically as a single bus, but a typical system has a number of buses such as a local bus and one or more expansion buses (e.g., ADB, SCSI, ISA, EISA, MCA, NuBus, or PCI), as well as serial and parallel ports. Network connections are usually established through a device such as a network adapter on one of these expansion buses or a modem on a serial port. The client computer may be a desktop system or a portable system.

The user interacts with the system using interface devices **37'** (or devices **37** in a standalone system). For example, client queries are entered via a keyboard, communicated to client processor **30'**, and thence to modem or network interface **40'** over bus subsystem **32'**. The query is then communicated to server **20** via network connection **45**. Similarly, results of the query are communicated from the server to the client via network connection **45** for output on one of devices **37'** (say a display or a printer), or may be stored on storage subsystem **35'**.

FIG. **1B** is a functional diagram of a computer system such as that of FIG. **1A**. FIG. **1B** depicts a server **20**, and a representative client **25** of a plurality of clients which may interact with the server **20** via the Internet **45** or any other communications method. Blocks to the right of the server are indicative of the processing steps and functions which occur in the server's program and data storage indicated by blocks **35***a* and **35***b* in FIG. **1A**. A TCP/IP "stack" **44** works in conjunction with Operating System **42** to communicate with processes over a network or serial connection attaching Server **20** to Internet **45**. Web server software **46** executes concurrently and cooperatively with other processes in server **20** to make data objects **50** and **51** available to requesting clients. A Common Gateway Interface (CGI) script **55** enables information from user clients to be acted upon by web server **46**, or other processes within server **20**. Responses to client queries may be returned to the clients in the form of a Hypertext Markup Language (HTML) document outputs which are then communicated via Internet **45** back to the user.

Client **25** in FIG. **1B** possesses software implementing functional processes operatively disposed in its program and data storage as indicated by block **35***a'* in FIG. **1A**. TCP/IP stack **44'**, works in conjunction with Operating System **42'** to communicate with processes over a network or serial connection attaching Client **25** to Internet **45**. Software implementing the function of a web browser **46'** executes concurrently and cooperatively with other processes in client **25** to make requests of server **20** for data objects **50** and **51**. The user of the client may interact via the web browser **46'** to make such queries of the server **20** via Internet **45** and to view responses from the server **20** via Internet **45** on the web browser **46'**.

Network Overview

FIG. **1C** is illustrative of the internetworking of a plurality of clients such as client **25** of FIGS. **1A** and **1B** and a plurality of servers such as server **20** of FIGS. **1A** and **1B** as described herein above. In FIG. **1C**, network **60** is an example of a prior art Token Ring or frame oriented network. Network **60** links host **61**, such as an IBM RS6000 RISC workstation, which may be running the AIX operating system, to host **62**, which is a personal computer, which may be running Windows, IBM OS/2 or a DOS operating system, and host **63**, which may be an IBM AS/400 computer, which may be running the OS/400 operating system. Network **60** is internetworked to network **70** via a system gateway which is depicted here as router **75**, but which may also be a gateway having a firewall or a network bridge. Network **70** is an example of an Ethernet network that interconnects host **71**, which is a SPARC workstation, which may be running SUNOS operating system with host **72**, which may be a VAX 6000 computer which may be running the VMS operating system (formerly available from Digital Equipment Corporation).

Router **75** is a network access point (NAP) of network **70** and network **60**. Router **75** employs a Token Ring adapter and Ethernet adapter. This enables router **75** to interface with the two heterogeneous networks. Router **75** is also aware of the Inter-network Protocols, such as ICMP and RIP, which are described herein below.

FIG. **1D** is illustrative of the constituents of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. The base layer of the TCP/IP protocol suite is the physical layer **80**, which defines the mechanical, electrical, functional and procedural standards for the physical transmission of data over communications media, such as, for example, the network connection **45** of FIG. **1A**. The physical layer may comprise electrical, mechanical or functional standards such as whether a network is packet switching or frame-switching; or whether a network is based on a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) or a frame relay paradigm.

Overlying the physical layer is the data link layer **82**. The data link layer provides the function and protocols to transfer data between network resources and to detect errors that may occur at the physical layer. Operating modes at the datalink layer comprise such standardized network topologies as IEEE 802.3 Ethernet, IEEE 802.5 Token Ring, ITU X.25, or serial (SLIP) protocols.

Network layer protocols **84** overlay the datalink layer and provide the means for establishing connections between networks. The standards of network layer protocols provide operational control procedures for internetworking communications and routing information through multiple heterogenous networks. Examples of network layer protocols are the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP). The Address Resolution Protocol (ARP) is used to correlate an Internet address and a Media Access Address (MAC) for a particular host. The Routing Information Protocol (RIP) is a dynamic routing protocol for passing routing information between hosts on networks. The Internet Control Message Protocol (ICMP) is an internal protocol for passing control messages between hosts on various networks. ICMP messages provide feedback about events in the network environment or can help determine if a path exists to a particular host in the network environment. The latter is called a "Ping". The Internet Protocol (IP) provides the basic mechanism for routing packets of information in the Internet. IP is a non-reliable communication protocol. It provides a "best efforts" delivery service and does not commit network resources to a particular transaction, nor does it perform retransmissions or give acknowledgments.

The transport layer protocols **86** provide end-to-end transport services across multiple heterogenous networks. The User Datagram Protocol (UDP) provides a connectionless, datagram oriented service which provides a non-reliable delivery mechanism for streams of information. The Transmission Control Protocol (TCP) provides a reliable session-based service for delivery of sequenced packets of information across the Internet. TCP provides a connection oriented reliable mechanism for information delivery.

The session, or application layer **88** provides a list of network applications and utilities, a few of which are illustrated here. For example, File Transfer Protocol (FTP) is

a standard TCP/IP protocol for transferring files from one machine to another. FTP clients establish sessions through TCP connections with FTP servers in order to obtain files. Telnet is a standard TCP/IP protocol for remote terminal connection. A Telnet client acts as a terminal emulator and establishes a connection using TCP as the transport mechanism with a Telnet server. The Simple Network Management Protocol (SNMP) is a standard for managing TCP/IP networks. SNMP tasks, called "agents", monitor network status parameters and transmit these status parameters to SNMP tasks called "managers." Managers track the status of associated networks. A Remote Procedure Call (RPC) is a programming interface which enables programs to invoke remote functions on server machines. The Hypertext Transfer Protocol (HTTP) facilitates the transfer of data objects across networks via a system of uniform resource indicators (URI).

The Hypertext Transfer Protocol is a simple protocol built on top of Transmission Control Protocol (TCP). It is the mechanism which underlies the function of the World Wide Web. The HTTP provides a method for users to obtain data objects from various hosts acting as servers on the Internet.

2.0 Traffic Class

A traffic class (or "class") is broadly defined as a grouping of traffic flows that share the same characteristics. A traffic class is defined with one or more matching rules. Traffic classes may have the property of being directional, i.e. all traffic flowing inbound will belong to different traffic classes and be managed separately from traffic flowing outbound. The directional property enables asymmetric classification and control of traffic, i.e., inbound and outbound flows belong to different classes which may be managed independent of one another.

Traffic classes may be defined at any level of the IP protocol as well as for other non-IP protocols. For example, at the IP level, traffic may be defined as only those flows between a specified set of inside and outside IP addresses or domain names. An example of such a low level traffic class definition would be all traffic between my network and other corporate offices throughout the Internet. At the application level, traffic classes may be defined for specific URIs within a web server. Traffic classes may be defined having "Web aware" class attributes. For example, a traffic class could be created such as all URIs matching "*.html" for all servers, or all URI patterns matching "*.gif" for server X, or for access to server Y with URI pattern "/sales/*" from client Z, wherein '*' is a wildcard character, i.e., a character which matches all other character combinations. Traffic class attributes left unspecified will simply match any value for that attribute. For example, a traffic class that accesses data objects within a certain directory path of a web server is specified by a URI pattern of the directory path to be managed, e.g. "/sales/*".

2.1 Classifying Traffic

The present invention provides a method for classifying traffic according to a definable set of classification attributes selectable by the manager, including selecting a subset of traffic of interest to be classified. The invention provides the ability to classify and search traffic based upon multiple orthogonal classification attributes.

Traffic class membership may be hierarchical. Thus, a flow may be classified by a series of steps through a traffic class tree, with the last step (i.e., at the leaves on the classification tree) mapping the flow to a policy. Some applications may be classified by application-specific attributes as well. For example, web traffic may also be classified by HTTP header types such as Content-Type (MIME type) or User-Agent. Citrix traffic may be classified by application name or Citrix client name. (A Citrix client name is the client name used under the Critix (brand) technique of client-server interaction. RTP (real-time protocol) traffic may be classified by encoding name or media type.

A classification tree is a data structure representing the hierarchical aspect of traffic class relationships. Each node of the classification tree represents a class, and has a traffic specification, i.e., a set of attributes or characteristics describing the traffic associated with it. Leaf nodes of the classification tree may contain policies. According to a particular embodiment, the classification process checks at each level if the flow being classified matches the attributes of a given traffic class. If it does, processing continues down to the links associated with that node in the tree. If it does not, the class at the level that matches determines the policy for the flow being classified. If no policy specific match is found, the flow is assigned the default policy.

In a preferred embodiment, the classification tree is an N-ary tree with its nodes ordered by specificity. For example, in classifying a particular flow in a classification tree ordered first by organizational departments, the attributes of the flow are compared with the traffic specification in each successive department node and if no match is found, then processing proceeds to the next subsequent department node. If no match is found, then the final compare is a default "match all" category. If, however, a match is found, then classification moves to the children of this department node. The child nodes may be ordered by an orthogonal paradigm such as, for example, "service type." Matching proceeds according to the order of specificity in the child nodes. Processing proceeds in this manner, traversing downward and from left to right in FIGS. 2A and 2B, which describe a classification tree, searching the plurality of orthogonal paradigms. Key to implementing this a hierarchy is that the nodes are arranged in decreasing order of specificity. This permits search to find the most specific class for the traffic before more general.

The table below depicts exemplary components from which traffic classes may be built. Note that the orientation of the server (inside or outside) may be specified. As noted above, any traffic class component may be unspecified, i.e., set to match any value.

Components of a Traffic Class Specifier

Inside (Client or Server)	Global	Outside (Server or Client)
IP Address/ Domain Name	TCP or UDP Service	IP Address/ Domain Name
Port Number	e.g., WWW,	Port Number
MAC Address	FTP, RealAudio, etc.	MAC Address
	URI pattern for Web Service,	
	MIME type for Web Service	
	IPX Service	
	SNA Service	
	LAT Service	
	IP precedence	
	Application Specific Attributes	

As an example, FIGS. 2A and 2B depict representative classifications of traffic made by a hypothetical network manager in order to accomplish particular allocations of

bandwidth. In FIG. 2A, the network manager has decided to divide her network resources first by allocating bandwidth between Departments A and B. FIG. 2A shows the resulting classification tree **201**, in which Department A bandwidth resources **202** and Department B bandwidth resources **204** each have their own nodes representing a specific traffic class for that department. Each traffic class may have one or more rules which are used for determining whether a flow matches that traffic class. For example, in FIG. 2A, the Department A resources node **202** has the matching rule Inside Host Subnet A associated with it. Next, the network manager has chosen to divide the bandwidth resources of Department A between two applications. She allocates an FTP traffic class **206** and a World Wide Web server traffic class **208**. Each of these nodes may have a separate policy attribute associated with them. For example, in FIG. 2A, the FTP node **206** for has an attribute Outside port **20** associated with it. Similarly, the network manager has chosen to divide network bandwidth resources of Departnent B into an FTP server traffic class **210** and a World Wide Web server traffic class **212**. Each may have their own respective policies.

FIG. 2B shows a second example **203**, wherein the network manager has chosen to first divide network bandwidth resource between web traffic and TCP traffic. She creates three traffic nodes, a web traffic node **220**, a TCP traffic node **224** and a default node **225**. Next, she divides the web traffic among two organizational departments by creating a Department A node **226**, and a Department B node **228**. Each may have its own associated policy. Similarly, she divides TCP network bandwidth into separate traffic classes by creating a Department A node **230** and a Department B node **232**. Each represents a separate traffic class which may have its own policy.

All traffic which does not match any user specified traffic class falls into an automatically created default traffic class which has a default policy. In FIG. 2A, the default category is depicted by a default node **205**, and in FIG. 2B, the default category is depicted by a default node **225**.

3.1 Traffic Discovery

Network traffic is classified under existing classes, beginning with the broadest classes, in inbound and outbound traffic classes which are protocol layer independent. For example, a particular instance of traffic may be classified according to its transport layer characteristics, e.g., Internet Protocol port number, as well as its application layer information, e.g., SMTP. In addition to application layer information, there may be attributes of the application upon which traffic classification can be based. For example, with SMTP, an attribute might be the length of the message or header information of the message or whether attachments are present. For an end-user application, such as a Lotus Notes database manager, the type of data might also be the basis of traffic classification. The key is that the flow specification must have some of its attributes discoverable in the course of the classification processing. Characteristics such as MIME types may also be discovered. Standard protocols, such as, IPX, SNA, and services, such as, SMTP and FTP are recognized for discovery. Classification is performed to the most specific level determinable. For example, in select embodiments, non-IP traffic, such as SNA, may be classified only by protocol, whereas within Internet Protocol, TCP or UDP traffic may be classified to the service level as indicated in the "/etc/services" file. Classification beyond a determined terminal classification level is not performed. For example, in a select embodiment, a class matching "ipx" or "nntp" will not be further classified.

3.1.1 Service Aggregates

A service aggregate is provided for certain applications that use more than one connection in a particular conversation between a client and a server. For example, an FTP client in conversation with an FTP server employs a command channel and a data transfer channel, which are distinct TCP sessions on two different ports. In cases where two or three TCP or UDP sessions exist for each conversation between one client and one server, it is useful to provide a common traffic class i.e., the service aggregate, containing the separate conversations. In practice, these types of conversations are often between the same two hosts, but using different ports. According to the invention, a class is created with a plurality of matching rules, each matching various component conversations.

3.1.2 Classification Under Specified Criterion

Classification of traffic into a tree is performed by traversing the tree of traffic classes, starting at the root and proceeding through each child of the root, comparing the flow being classified against the matching rules associated with each traffic class. The flow is defined as "matching" a class if its characteristics match any one of the matching rules that is used to define the class. When the flow matches a class, then if that traffic class has children, the flow will be compared against each of the children to determine if there is a more specific match—otherwise, the processing stops and the flow is assigned to that traffic class. A marker is placed in match_all default nodes so that when match processing reaches the marker, the classification processing depicted in flowchart **403** terminates, and the flow is assigned to the default class that was reached.

3.1.3 Default Suggested Policies

A default policy may be suggested or, in select embodiments, automatically applied, to a traffic class which has been discovered. Applying suggested or default policies for a new class at a user's option is described in a copending, commonly owned, U.S. patent application Ser. No. 09/198,051, entitled, "Method for Automatically Determining a Traffic Policy in a Packet Communications Network," which is incorporated herein by reference in its entirety for all purposes.

3.1.4 Analysis of Data in Determining Traffic Class

In a preferable embodiment, classification can extend to examination of the data contained in a flow's packets. Certain traffic may be distinguished by a signature even if it originates with a server run on a non-standard port, for example, an HTTP conversation on port 8080 would not be otherwise determinable as HTTP from the port number. Further analysis of the data is conducted in order to determine classification in instances where: 1) FTP commands are used to define server ports, 2) HTTP protocol is used for non-web purposes. The data is examined for indication of push traffic, such as Pointcast Network-type traffic (a type of traffic marketed by InfoGate of San Diego, Calif.), which uses HTTP as a transport mechanism. These uses may be isolated and classified into a separate class. Marimba and Pointcast can be distinguished by looking into the data for a signature content header in the get request. Pointcast has URLs that begin with "/FIDO-1/." Other applications in which protocol can be inferred from data include Telnet traffic. Both tn3270 and tn3270E (emulation) may be detected by looking into data and given a different class. Telnet traffic has option negotiations which may indicate an appropriate class.

3.1.5 Identity of Traffic Based Upon Resource Creator's Class

A flow's traffic class may be inferred by observing the existence of other flows that are known to be related, between the same two nodes as the flow being classified. This method is used to detect Real Time Protocol (RTP) for point-to-point telephony, RTP for broadcast streaming, CCITT/ITU, H323-internet telephony over the Internet (bidirectional) and RTSP real time streaming protocol (unidirectional).

3.1.6 Dynamic Ports

Some applications establish connections to a well-known port number. Other applications use dynamic ports, by first connecting to a well-known port number, and then being redirected to another port number which is random or dynamically generated. For example, in a database application, a client may connect to the database server's well-known port number. At this location, a load-balancing server may be running which is aware of all of the other port numbers that are listened to by instances of the database application. The load-balancing server will redirect the client to the port number of the least-loaded database server instance.

3.2 Traffic Discovery Processing

FIG. **3** depicts components of a system for discovering traffic according to the invention. A traffic tree **302** is provided in which new traffic will be discovered under a particular member class node. The traffic tree may have a hierarchy of nodes (Class A, B C) and corresponding sub-nodes under the nodes. A traffic classifier **304** detects services for incoming traffic. Alternatively, the classifier may start with a service and determine the hosts using it. A knowledge base **306** contains heuristics for determining traffic classes. The knowledge base may be embodied in memory, file, executable code, or a database. In a preferred embodiment, the knowledge is contained within a data structure resident in memory, and in executable code. A plurality of saved lists **308** stores identifying characteristics of classified traffic pending incorporation into traffic tree **302**. In select embodiments, entries for each instance of traffic may be kept in one of the plurality of saved lists, each of which is associated with a traffic class which is marked to indicate that discovery is enabled on it. If there are attributes that are specific to an application, the entries may contain such attributes. In alternate embodiments, a copy of an entry and a count of duplicate copies for the entry is maintained.

FIG. **4**A depicts a flowchart **401** of processing steps for discovering traffic. In a step **402**, a flow specification is parsed from the flow being classified. The flow specification may include attributes of the application associated with the flow. Then in a step **404**, the flow specification parsed from the flow in step **402** is compared with the traffic specifications in each node of the classification tree. Rules are checked starting from most specific to least specific. In a decisional step **406**, a determination is made if traffic matches one of the classes that are marked for discovery. If this is so, then in a step **408**, an entry is made in a list of identifying characteristics, such as protocol type, IP protocol number, server port, traffic type if known, application-specific attributes, or a time of occurrence of the traffic. In an optional step **410**, duplicate instances having the same identifying characteristics are suppressed, in favor of keeping a count of the duplicates and a most recent time traffic with these identifying characteristics was encountered. In an optional step **412**, a byte count of traffic of this type has been detected is included. It should be noted that as a result of the traffic classification process, a flow will always match some traffic class. If it does not match anything specific, it will match a match-all (default) class. Also, it should be noted that if a flow matches a class that is marked for discovery, information about the flow will always be recorded in the list of saved characteristics.

In a preferred embodiment, processing according to flowchart **401** may execute on multiple instances of saved list **308**.

3.2.1 Displaying Results to a User

In an optional step **413** (not shown), after the processing of flowchart **401** completes or at periodic intervals or on demand, a list of traffic classes produced in steps **402** through **412** is displayed to a network manager. The list may be sorted by any well-known criteria such as: 1) most "hits" during a recent interval, 2) most recently-seen (most recent time first), 3) most data transferred (bytes/second) during some interval, or a moving average. The user may choose an interval length or display cutoff point (how many items, how recent, at least B bytes per second, or other thresholds). The Network manager may then take some action (e.g. pushing a button) to select the traffic types she wishes to add to the classification tree. The display can be hierarchical, as depicted in lines (3) below:

FTP (3)
FTP-cmd
FTP-data
Lotus
Lotus_database **1**
Lotus_database **2**
Lotus_database **2**_video
to-host1
tcp
FTP
FTP-cmd
FTP-data
HTTP
images
java
text
TCP-port-9999

wherein the "TCP-port-9999" entry is a traffic class which was discovered as a result of an application which was making repeated or simultaneous connections to TCP port 9999, and for which there was no other information available to allow matching on a specific class for that application (it was not an application known in the knowledge base.)

The italicized terms are examples of sub-nodes with application-specific characteristics.

In a related embodiment, a threshold for display or class creation of well-known traffic types is provided.

3.2.2 Interval Based Incorporation

In an alternative embodiment, at select intervals of time, items in the saved list of traffic characteristics are analyzed, and either 1) recognized and a corresponding traffic class is added to the tree, or 2) (for repeated attempts to request a server connection port, IP subprotocol type, or ethertype that is not otherwise known in the knowledge base, upon exceeding a certain threshold) a class for the traffic is created and added to the classification tree.

FIG. **4**B depicts a flowchart **403** of the processing steps for integrating traffic classes into a classification tree in an

embodiment. Processing steps of flowchart **403** create new classes in the classification tree, based upon the information saved in the lists of traffic characteristics. This is done periodically at a defined interval of seconds, such as 30 second in a preferred embodiment. In a step **420**, an instance of saved traffic is retrieved from the saved traffic list **308**. Thereafter, a test is made to determine if discovery is occurring under a class for which discovery of attributes occurs (that is, a class which corresponds to an application which has application-specific attributes which may be discovered). (step **421**). If true, then the process proceeds directly to the step of creating a node for a traffic class for the next attribute that was saved (step **424**). Elements on the list of discoverable attributes are assigned a processing priority relative to one another. It is a characteristic of the product that the processing priority ("ordering") may be pre-defined or may be configurable by the user. (While the characteristic of order is a part of the invention, the mechanism for fixing or for reconfiguring the ordering is not a part of this invention.) If the process is not in a state of discovering attributes, then in an alternative decisional step **422**, the instance of saved traffic is examined to determine whether it is well-known (e.g., protocol type, assigned port number) and a name representing its type exists. If this is so then processing continues with a test of whether the saved traffic belongs to a service aggregate in step **426**. Otherwise, in a step **423** the instance of saved traffic is examined to determine whether it appears to be a server connection port of an unregistered IP port (or a port that has not been configured). If this is not so then, processing continues with the next traffic class in the saved list in step **420**. In decisional step **426**, the instance of saved traffic is examined to determine whether it belongs to a service aggregate. For example, an FTP session has one flow that is used to exchange commands and responses and a second flow that is used to transport data files. If the traffic does belong to a service aggregate, then in a step **428**, a traffic class is created which will match all components of the service aggregate. In a step **425**, a new traffic class is created to match the instance of saved traffic.

In an optional step (not shown), a suggested policy is determined for the traffic class created in step **425**. Next, in a decisional step **432**, a limit is checked to verify that the total number of traffic classes has not exceeded a specified maximum. If the limit on classes has not been reached, then the traffic is checked to determine if there are still attributes for which classes have not been discovered (step **434**) and if so, then the attributes are retained in the list (step **436**). In either case, the process is repeated from step **420**.

In a related embodiment in place of steps **424**, **425** or **428**, a display of traffic classes, sorted by most recently used, most hits, number of bytes received during any interval, which is determined by a plurality of time stamps, is available on demand to a network manager. The network manager then manually indicates that the traffic is to be added to the tree.

In a particular embodiment a threshold is employed to determine traffic for which a separate class should be added. A minimum usage threshold indicates whether a particular port has been used at least n times in the last s seconds. (This applies only in those instances where there is an identifiable port.) If traffic is well known, i.e., SMTP, a new traffic class is created immediately, i.e., threshold is equal to one hit per minute; otherwise, the threshold is set equal to a fixed, arbitrarily-configured value, for example, two to ten thousand hits per minute. A new class for traffic is given a generic name, e.g., TCP-Port-99. Entries for traffic over a certain age, for example one minute old, are discarded.

In a related embodiment, another method of identifying an individual traffic class is to detect simultaneous connections to the same host port from different clients. This provides an indication that the port is a well-known connection port.

Traffic classes are created for any combination of the above mentioned categories. A flag is added to all traffic classes so created in order to indicate that it is the product of the auto classifier.

3.2 Command Language Interface:

In a particular embodiment, function of the classifier **304** is controlled by a command language interface. Below is a plurality of command language interface commands.

setup discover {on|off} To activate autoclassification for various classes to detect well-known protocols and services:

class discover <tclass>{inside/outside/both} To turn on autoclassification (a.k.a. discovery) under a class to detect services with the host on the inside, the outside, or both directions.

class discover <tclass> off To turn off use.

The new classes have names in the format of lines below:

<service> or

<protocol>_Port_<number> or

<service>_<attribute>[_<attribute>_<attribute> . . .]

where <protocol> is either TCP or UDP.

If a heretofore unknown server-connection port appears to be "well used", an entry of the second type is created. The threshold for creation is for example 11 hits with no more than 1 minute (granularity of checking is at least 30 seconds between running successive discovery or autoclassification processes) between any two hits. For example:

inbound/inside/ftp

http

sna

TCP_port_505

Lotus_database2_video

3.3 Syntax of Traffic Specifications (a.k.a. Matching Rules):

Flow specifications and traffic specifications may have an inside service field and an outside service field. (For some protocols or service types, inside and outside are not distinguished.) Each will have values of SVC_UNKNOWN (0), SVC_CLIENT (1), or a number greater than 1, which is the service index, an index into the global table gServiceTable. If a type of service is known for a connection, the service field at a particular side will be set to SVC_CLIENT and the service field at the opposite side will be the index into gServiceTable. If a type of service is not known for the traffic, both inside service field and outside service field will be SVC_UNKNOWN. A person of reasonable skill in the art will appreciate that other embodiments for the table, such as representing the information contained therein as text strings or by any one of a plurality of possible encoding schemes, are realizable without departing from the present invention.

Therefore, a traffic specification can have "outside service:http" (or just "outside HTTP") which is different than "outside tcp port:**80**". The first will match HTTP on any port while the second will match anything on port **80** (including Pointcast and Marimba).

Specifying an aggregate traffic specification "service:<agg>" identifies the traffic specifications for various traffic belonging to the service. Specifying "class new

inbound cdls outside dls" is the same as "class new inbound cdls outside service:dls-wpn" and "class tspec add cdls outside service:dls-rpn". Most services which are known in the knowledge base will create a class that encompasses all of the members of the aggregate.

Network managers need not be aware of services which are known to be derivative of others, e.g., Pointcast and Marimba are special cases of HTTP and tn3270 is a special case of Telnet, in order to work with the system.

4.0 Conclusion

In conclusion, the present invention provides for an automatic determination of a policy for a packet telecommunications system wherein bandwidth is allocated to requesting flows according to automatically determined application requirements. An advantage of traffic classification techniques according to the present invention is that network managers need not know the technical aspects of each kind of traffic in order to configure traffic classes. A further advantage of the present invention is that traffic classes may include application-specific attributes such as a MIME type for web traffic.

Other embodiments of the present invention and its individual components will become readily apparent to those skilled in the art from the foregoing detailed description. As will be realized, the invention is capable of other and different embodiments, and its several details are capable of modifications in various obvious respects, all without departing from the spirit and the scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not as restrictive. It is therefore not intended that the invention be limited except as indicated by the appended claims.

What is claimed is:

1. A method for automatically classifying traffic in a packet communications network, said network having any number of flows, including zero, comprising the steps of:

parsing a packet into a first flow specification, wherein said first flow specification contains at least one instance of any one of the following:

a protocol family designation,

a direction of packet flow designation,

a protocol type designation,

a pair of hosts,

a pair of ports,

in HTTP protocol packets, a pointer to a MIME type; thereupon,

matching the first flow specification of the parsing step to a plurality of classes represented by a plurality of nodes, each node having a traffic specification; thereupon,

if a matching node was not found in the matching step, associating said first flow specification with one or more newly-created nodes; thereupon,

incorporating said newly-created node into said plurality of nodes.

2. A method for automatically classifying traffic in a packet communications network, said network having any number of flows, including zero, comprising the steps of:

determining application type of a flow; thereafter

for said application type of said flow, parsing a packet of said flow into a first flow specification, said first flow specification containing information as attributes, said attributes being specific to said application type and wherein selected ones of said attributes are discoverable; thereupon

matching the first flow specification of the parsing step to a plurality of classes represented by a plurality of nodes of a classification tree type, each said classification tree type node having a traffic specification; thereupon

if a matching classification tree type node was found in the matching step and said matching classification tree type node indicates that further nodes can be created as a consequence of attributes thereof that are discoverable, then

creating at least one new classification tree type node; thereupon

associating said first flow specification with said at least one newly-created classification tree type node; and thereupon

incorporating said at least one newly-created classification tree type node into said plurality of classification tree type nodes so that policies can be applied to traffic based only on said discoverable attributes of said at least one newly-created classification tree type node.

3. The method of claim 2 wherein said discoverable attributes are assigned a processing priority relative to one another.

4. The method of claim 2 further comprising the steps of:

for at least a second flow having a second flow specification, recognizing said second flow specification and said first flow specification to comprise together a service aggregate; thereupon

associating said first flow specification and said second flow specification with a newly-created classification tree node, said newly-created classification tree type node having a first traffic specification corresponding to said first flow specification and a second traffic specification corresponding to said second flow specification.

5. The method of claim 2 farther comprising the steps of:

applying policies from said newly-created classification tree type nodes to instances of detected traffic.

6. The method of claim 2 further comprising the steps of:

for a subclassification under a specified criterion comprising a specified attribute name and a value, if a matching classification tree type node was found in the matching step, said matching classification tree type node having at least one child classification tree type node, applying the matching, associating, and incorporating steps to a particular child classification tree type node of said matching classification tree type node as a part of classification.

7. The method of claim 2 wherein the parsing step further comprises the steps of:

examining data contained within a plurality of component packets belonging to said first flow for any number of a plurality of indicators of any of the following:

a protocol;

a service; thereupon, matching said plurality of indicators to said classes represented by a plurality of said classification tree type nodes.

8. The method of claim 2 further including measuring traffic load and invoking said classification upon achievement of a minimum usage threshold.

9. The method according to claim 2 wherein said matching step is applied to hierarchically-recognized classes.

10. A system for automatically classifying traffic in a packet telecommunications network, said network having any number of flows, including zero, comprising:

a plurality of network links upon which said traffic is carried;

a network routing means; and

a processor means operative to:

determine application type of a flow;

for said application type of said flow, parse a packet of said flow into a first flow specification, said first flow specification containing information as attributes, said attributes being specific to said application type and wherein selected ones of said attributes are discoverable; thereupon

match the first flow specification of the parsing step to a plurality of classes represented by a plurality of nodes of a classification tree type, each said classification tree type node having a traffic specification; thereupon

if a matching classification tree type node was found in the matching step and said matching classification tree type node indicates that further nodes can be created as a consequence of attributes thereof that are discoverable, then

associate said first flow specification with said at least one newly-created classification tree type node; thereupon

create at least one new classification tree type node; and thereupon

incorporate said at least one newly-created classification tree type node into said plurality of classification tree type nodes so that policies can be applied to traffic based only on said discoverable attributes of said at least one newly-created classification tree type node.

11. The system of claim **10** wherein said processor means is further operative to include measuring traffic load and invoking said classification upon achievement of a minimum usage threshold.

12. The system according to claim **10** wherein said processor means is further operative to apply said matching step to hierarchically-recognized classes.

13. A method for classifying traffic in a packet telecommunications network, said network having any number of flows, including zero, said method comprising the steps of:

parsing a packet into a first flow specification, said first flow specification having discoverable attributes; thereupon,

matching the first flow specification of the parsing step to a plurality of classes represented by a plurality of classification tree type nodes, each said classification tree type node having a traffic specification; thereupon,

if a matching classification tree type node was found in the matching step and said matching classification tree type node indicates through said discoverable attributes that further nodes can be created, creating at least one new classification tree type node; thereupon

associating said first flow specification with at least one more newly-created node; thereupon,

displaying to a network administrator a representation of traffic according to said traffic specification for use in manual intervention.

14. The method according to claim **13** further including the step of sorting said traffic representation according to most recently occurring.

15. The method according to claim **13** further including the step of sorting said traffic representation according to most data transferred for a preselected period of time.

16. The method of claim **13** further including measuring traffic load and invoking said classification upon achievement of a minimum usage threshold.

17. The method according to claim **13** wherein said matching step is applied to hierarchically-recognized classes.

* * * * *

US006412000B1

(12) **United States Patent**
Riddle et al.

(10) **Patent No.: US 6,412,000 B1**
(45) **Date of Patent: Jun. 25, 2002**

(54) **METHOD FOR AUTOMATICALLY CLASSIFYING TRAFFIC IN A PACKET COMMUNICATIONS NETWORK**

(75) Inventors: **Guy Riddle**; **Robert L. Packer**, both of Los Gatos, CA (US)

(73) Assignee: **Packeteer, Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/198,090**

(22) Filed: **Nov. 23, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/066,864, filed on Nov. 25, 1997.

(51) **Int. Cl.**[7] **G06F 15/173**

(52) **U.S. Cl.** **709/224**; 709/223; 709/230; 709/238; 709/242; 370/230; 370/235; 370/252; 370/355; 370/356

(58) **Field of Search** 709/223–226, 709/230, 235–236, 238–239, 242, 246; 370/229–230, 235, 252–253, 355–356, 401, 466–469

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,251,152 A * 10/1993 Noteas 709/224
5,495,426 A * 2/1996 Waclawsky et al. 709/226
5,838,919 A * 11/1998 Schwaller et al. 709/224
5,870,561 A * 2/1999 Jarvis et al. 709/238
5,903,559 A * 5/1999 Acharya et al. 709/236
5,923,849 A * 7/1999 Venkatraman 709/224
6,028,842 A * 2/2000 Chapman et al. 370/252
6,046,980 A * 4/2000 Packer 370/230
6,137,782 A * 10/2000 Sharon et al. 709/238
6,209,033 B1 * 3/2001 Datta et al. 709/224

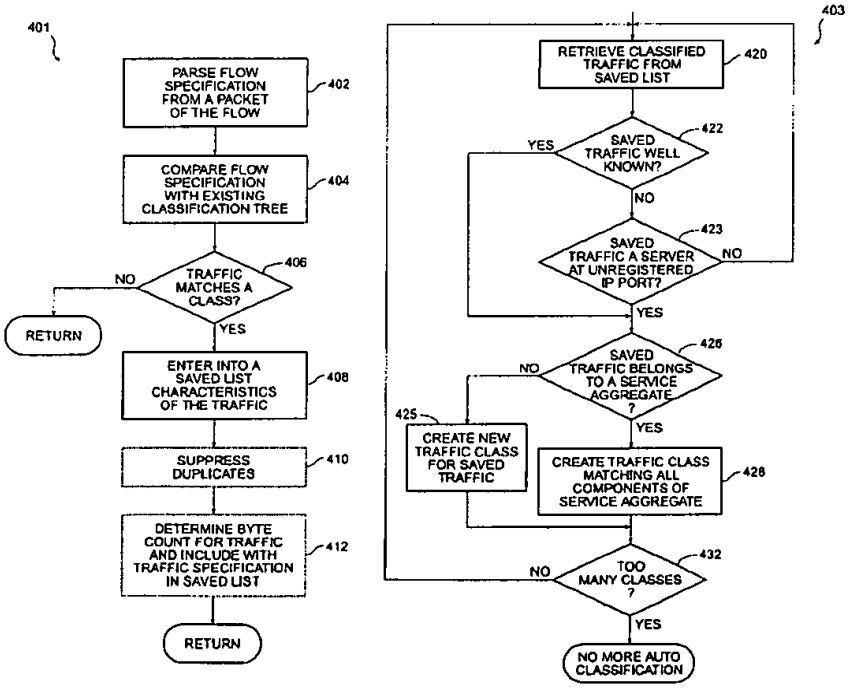
* cited by examiner

Primary Examiner—Zarni Maung
Assistant Examiner—Bharat Barot
(74) *Attorney, Agent, or Firm*—Townsend and Townsend and Crew LLP; Kenneth R. Allen

(57) **ABSTRACT**

In a packet communication environment, a method is provided for automatically classifying packet flows for use in allocating bandwidth resources by a rule of assignment of a service level. The method comprises applying individual instances of traffic classification paradigms to packet network flows based on selectable information obtained from a plurality of layers of a multi-layered communication protocol in order to define a characteristic class, then mapping the flow to the defined traffic class. It is useful to note that the automatic classification is sufficiently robust to classify a complete enumeration of the possible traffic.

15 Claims, 7 Drawing Sheets



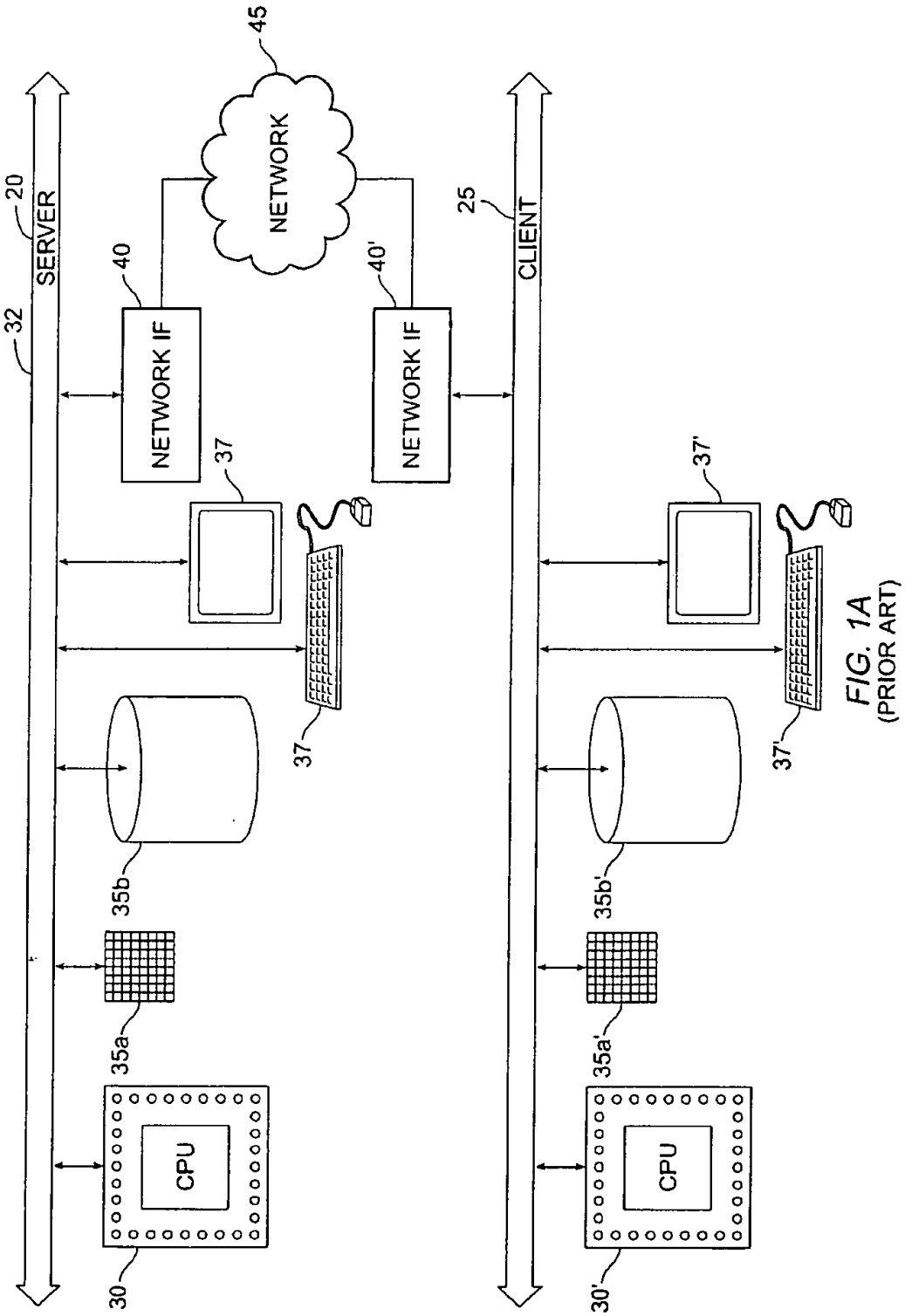


FIG. 1A
(PRIOR ART)

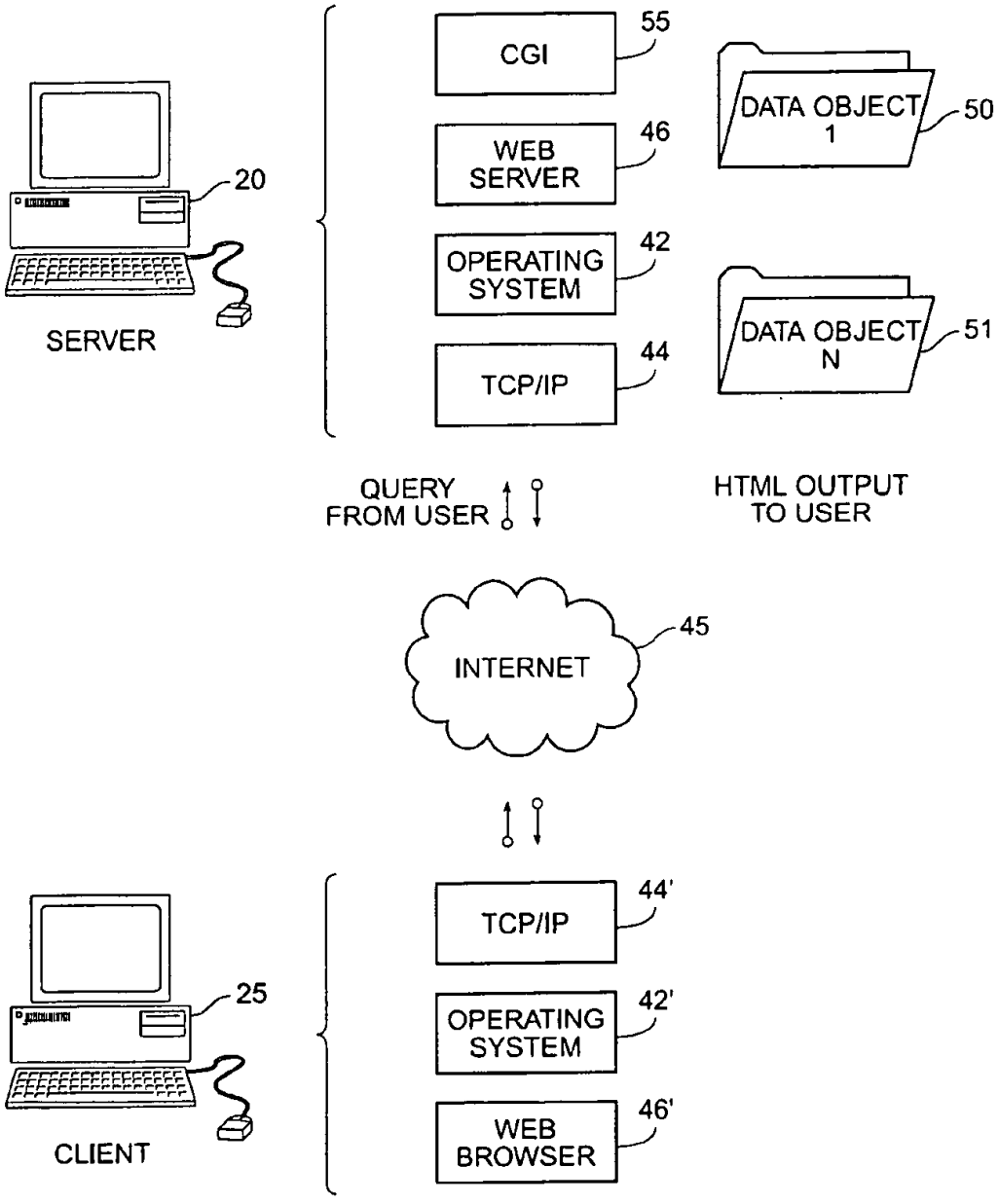


FIG. 1B
(PRIOR ART)

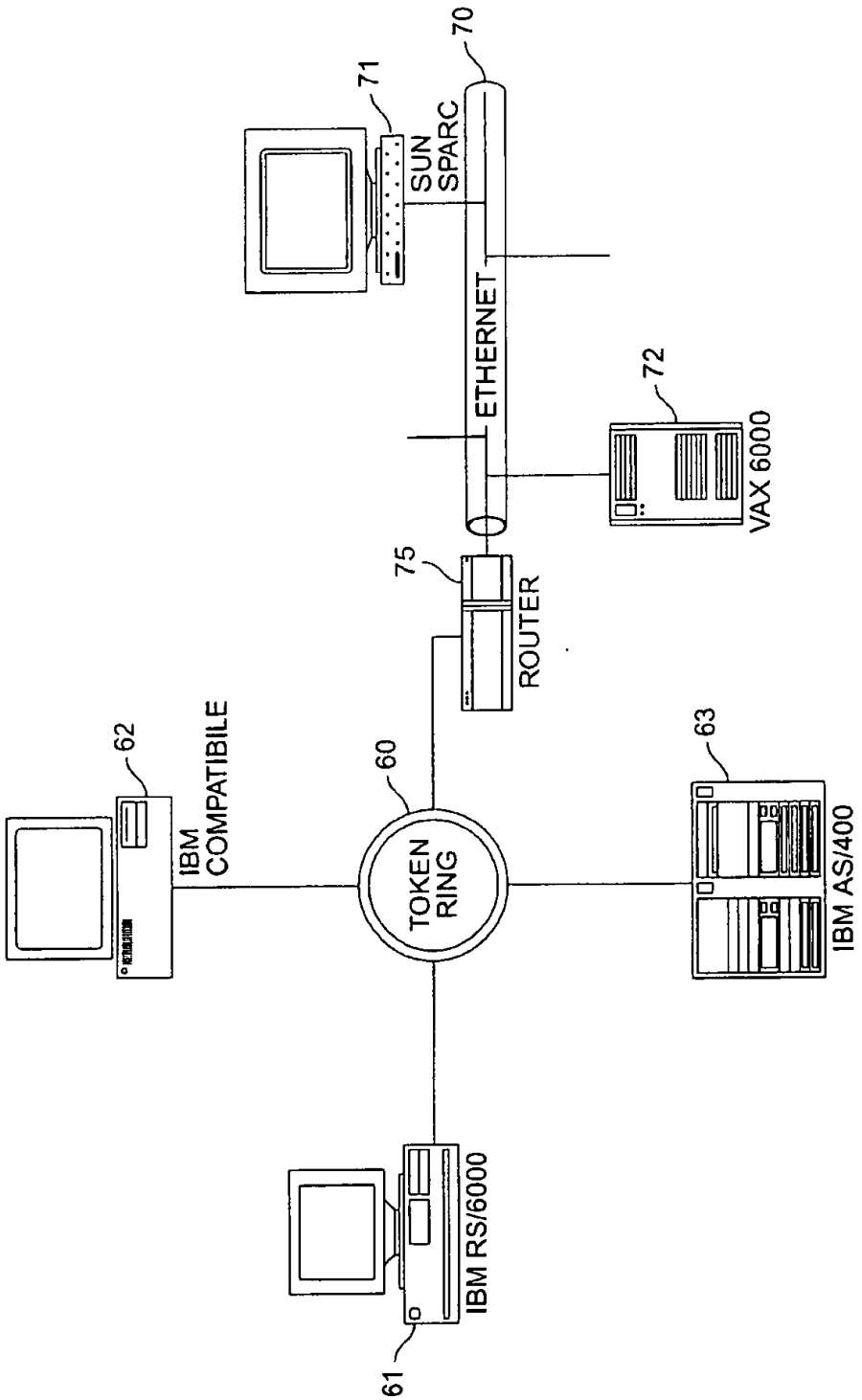
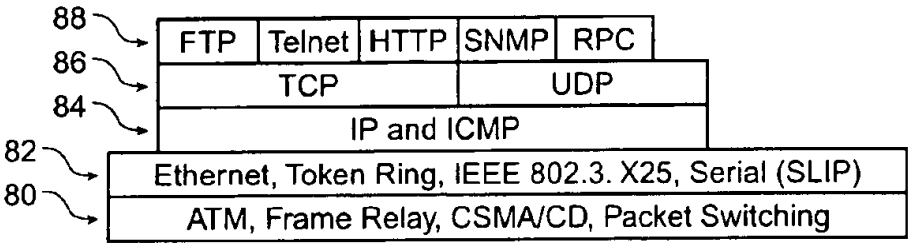


FIG. 1C
(PRIOR ART)



LEGEND

88 Session/Application Layer
86 Transport Layer
84 Network Layer
82 Data Link Layer
80 Physical Layer

FIG. 1D
(PRIOR ART)

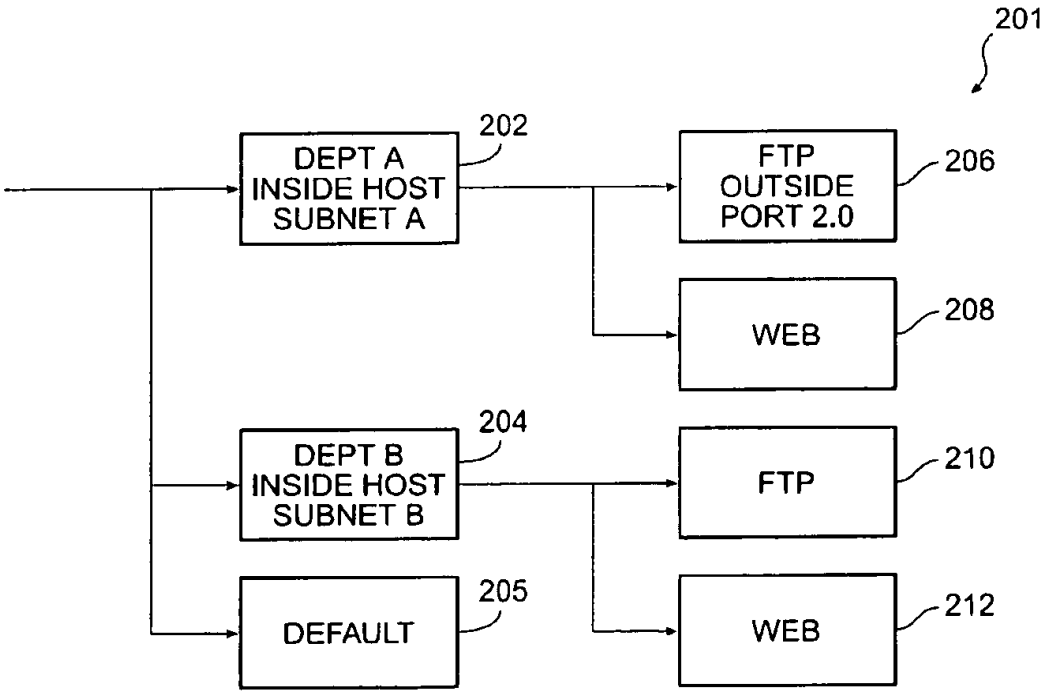


FIG. 2A

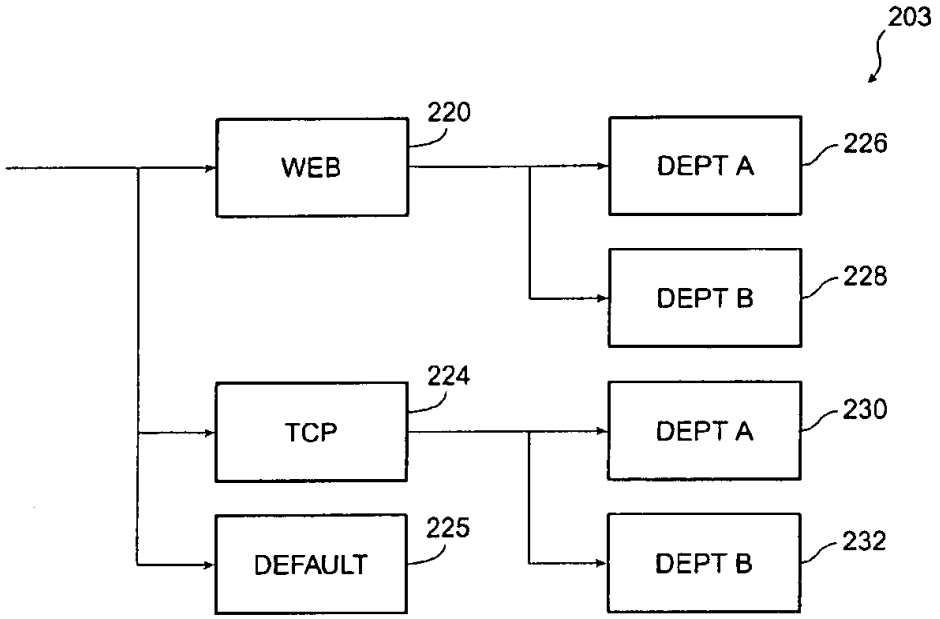

203
220
WEB
DEPT A
226
DEPT B
228
224
TCP
DEPT A
230
225
DEFAULT
DEPT B
232

FIG. 2B

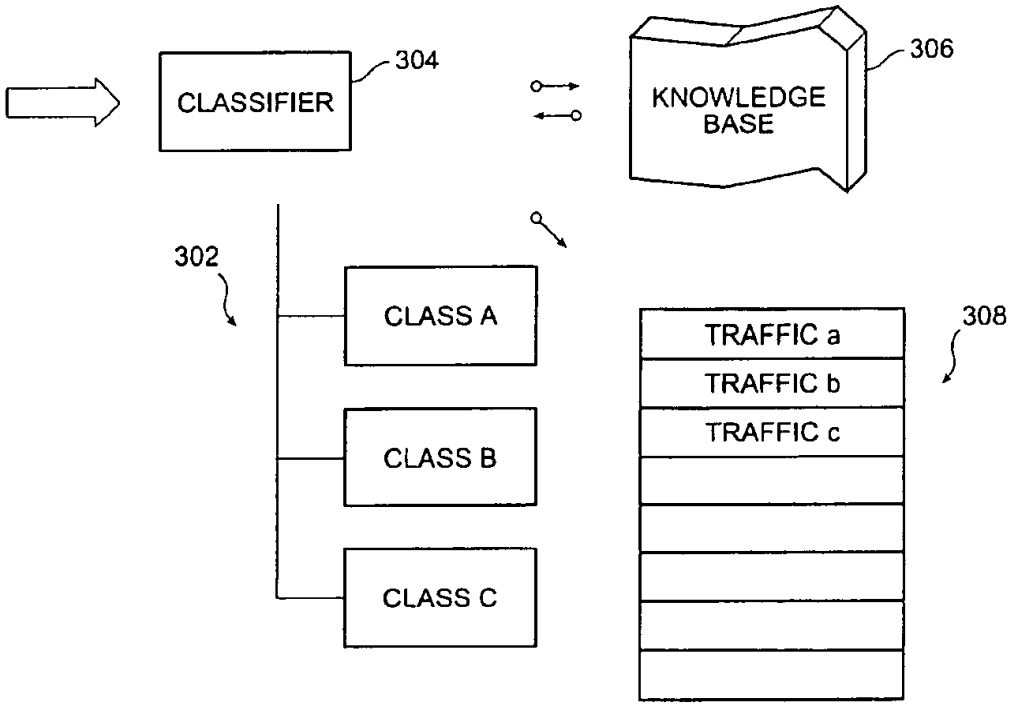

CLASSIFIER
304
KNOWLEDGE BASE
306
302
CLASS A
CLASS B
CLASS C
TRAFFIC a
TRAFFIC b
TRAFFIC c
308

FIG. 3

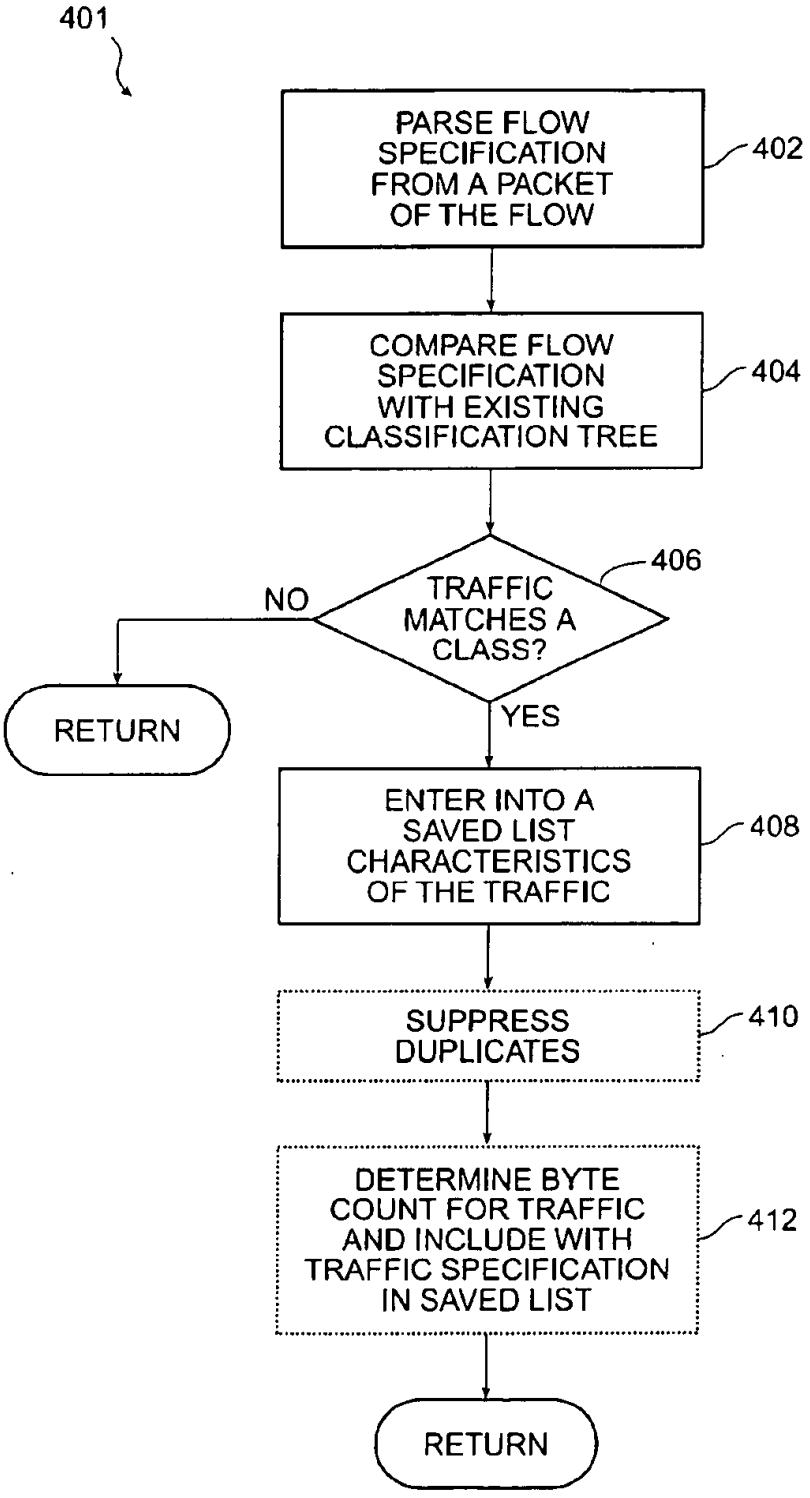

401
PARSE FLOW SPECIFICATION FROM A PACKET OF THE FLOW
402
COMPARE FLOW SPECIFICATION WITH EXISTING CLASSIFICATION TREE
404
TRAFFIC MATCHES A CLASS?
406
NO
YES
RETURN
ENTER INTO A SAVED LIST CHARACTERISTICS OF THE TRAFFIC
408
SUPPRESS DUPLICATES
410
DETERMINE BYTE COUNT FOR TRAFFIC AND INCLUDE WITH TRAFFIC SPECIFICATION IN SAVED LIST
412
RETURN

FIG. 4A

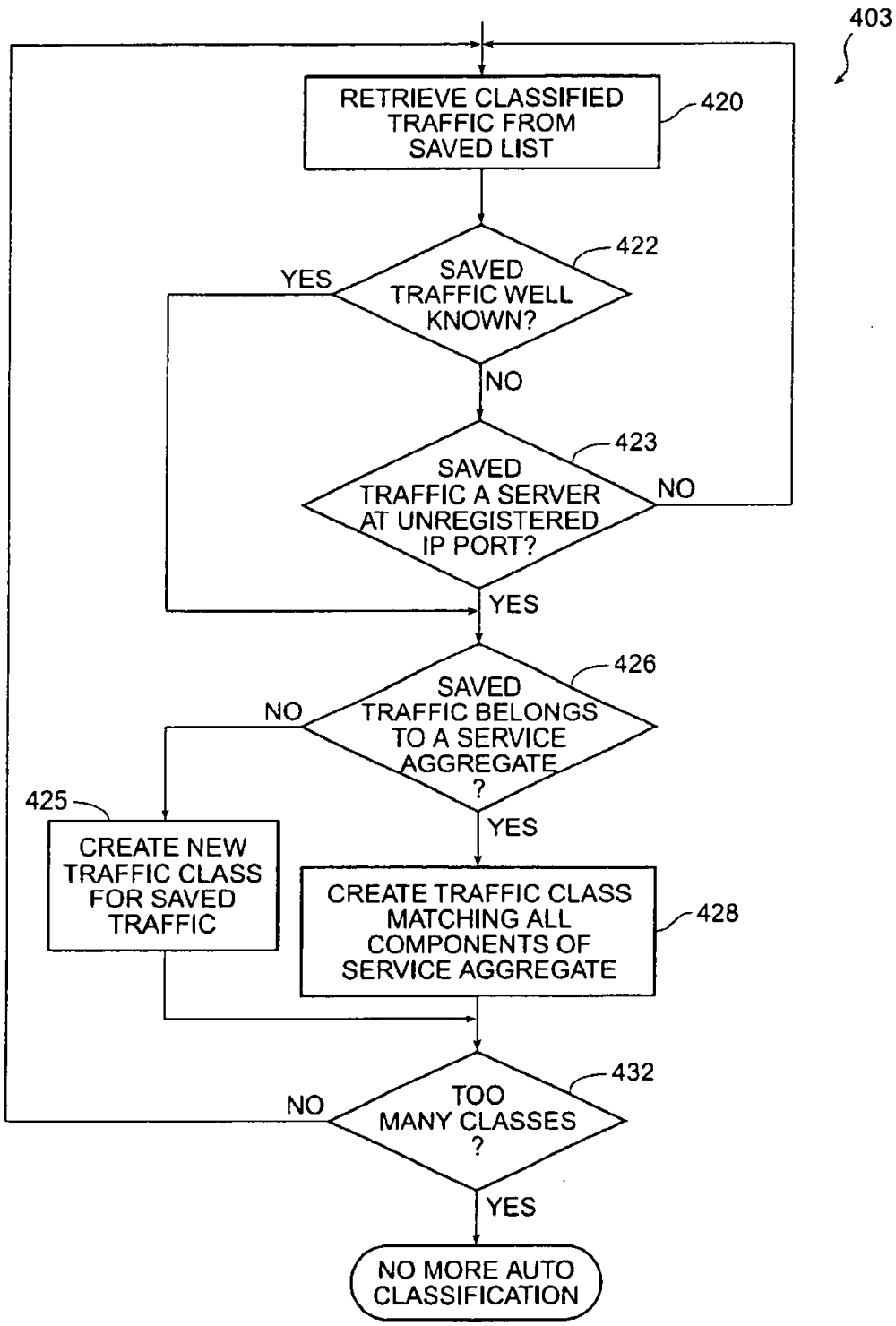

403
RETRIEVE CLASSIFIED TRAFFIC FROM SAVED LIST
420
SAVED TRAFFIC WELL KNOWN?
422
YES
NO
SAVED TRAFFIC A SERVER AT UNREGISTERED IP PORT?
423
NO
YES
SAVED TRAFFIC BELONGS TO A SERVICE AGGREGATE ?
426
NO
YES
425
CREATE NEW TRAFFIC CLASS FOR SAVED TRAFFIC
CREATE TRAFFIC CLASS MATCHING ALL COMPONENTS OF SERVICE AGGREGATE
428
TOO MANY CLASSES ?
432
NO
YES
NO MORE AUTO CLASSIFICATION

FIG. 4B

METHOD FOR AUTOMATICALLY CLASSIFYING TRAFFIC IN A PACKET COMMUNICATIONS NETWORK

CROSS-REFERENCES TO RELATED APPLICATIONS

This application claims priority from a commonly owned U.S. Provisional Patent Application, Ser. No. 60/066,864, filed on Nov. 25 1997, in the name of Guy Riddle and Robert L. Packer, entitled "Method for Automatically Classifying Traffic in a Policy Based Bandwidth Allocation System."

The following related commonly-owned contemporaneously-filed co-pending U.S. Patent Application is hereby incorporated by reference in its entirety for all purposes: U.S. patent application Ser. No. 09/198,051, still pending, in the name of Guy Riddle, entitled "Method for Automatically Determining a Traffic Policy in a Packet Communications Network,".

COPYRIGHT NOTICE

Further, this application makes reference to the following commonly owned U.S. Patent Application, which are incorporated by reference herein in their entirety for all purposes:

- U.S. Pat. No. 5,802,106, in the name of Robert L. Packer, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision," relates to a technique for automatically determining the data rate of a TCP connection;
- U.S. patent application Ser. No. 08/977,376, now U.S. Pat. No. 6,046,980, in the name of Robert L. Packer, entitled "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers in Store and Forward Network," relates to a technique for automatically allocating bandwidth based upon data rates of TCP connections according to a hierarchical classification paradigm; and.
- U.S. patent application Ser. No. 08/742,994, now U.S. Pat. No. 6,038,216 in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Environment Without a Data Rate Supervision," relates to a technique for automatically scheduling TCP packets for transmission.

BACKGROUND OF THE INVENTION

This invention relates to digital packet telecommunications, and particularly to management of network bandwidth based on information ascertainable from multiple layers of OSI network model. It is particularly useful in conjunction with bandwidth allocation mechanisms employing traffic classification in a digitally-switched packet telecommunications environment, as well as in monitoriing, security and routing.

The ubiquitous TCP/IP protocol suite, which implements the world-wide data communication network environment called the Internet and is also used in private networks (Intranets), intentionally omits explicit supervisory function over the rate of data transport over the various media which comprise the network. While there are certain perceived advantages, this characteristic has the consequence of juxtaposing very high-speed packet flows and very low-speed packet flows in potential conflict for network resources, which results in inefficiencies. Certain pathological loading conditions can result in instability, overloading and data transfer stoppage. Therefore, it is desirable to provide some mechanism to optimize efficiency of data transfer while minimizing the risk of data loss. Early indication of the rate of data flow which can or must be supported is imperative. In fact, data flow rate capacity information is a key factor for use in resource allocation decisions. For example, if a particular path is inadequate to accommodate a high rate of data flow, an alternative route can be sought out.

Internet/Intranet technology is based largely on the TCP/IP protocol suite, where IP, or Internet Protocol, is the network layer protocol and TCP, or Transmission Control Protocol, is the transport layer protocol. At the network level, IP provides a "datagram"delivery service. By contrast, TCP builds a transport level service over the datagram service to provide guaranteed, sequential delivery of a byte stream between two IP hosts.

TCP flow control mechanisms operate exclusively at the end stations to limit the rate at which TCP endpoints emit data. However, TCP lacks explicit data rate control. The basic flow control mechanism is a sliding window, superimposed on a range of bytes beyond the last explicitly-acknowledged byte. Its sliding operation limits the amount of unacknowledged transmissible data that a TCP endpoint can emit.

Another flow control mechanism is a congestion window, which is a refinement of the sliding window scheme, which employs conservative expansion to fully utilize all of the allowable window. A component of this mechanism is sometimes referred to as "slow start".

The sliding window flow control mechanism works in conjunction with the Retransmit Timeout Mechanism (RTO), which is a timeout to prompt a retransmission of unacknowledged data. The timeout length is based on a running average of the Round Trip Time (RTT) for acknowledgment receipt, i.e. if an acknowledgment is not received within (typically) the smoothed RTT+4*mean deviation, then packet loss is inferred and the data pending acknowledgment is retransmitted.

Data rate flow control mechanisms which are operative end-to-end without explicit data rate control draw a strong inference of congestion from packet loss (inferred, typically, by RTO). TCP end systems, for example, will "back-off", i.e., inhibit transmission in increasing multiples of the base RTT average as a reaction to consecutive packet loss.

Bandwidth Management in TCP/IP Networks

Conventional bandwidth management in TCP/IP networks is accomplished by a combination of TCP end systems and routers which queue packets and discard packets when certain congestion thresholds are exceeded. The discarded, and therefore unacknowledged, packet serves as a feedback mechanism to the TCP transmitter. (TCP end systems are clients or servers running the TCP transport protocol, typically as part of their operating system.) The term "bandwidth management" is often used to refer to link level bandwidth management, e.g. multiple line support for Point to Point Protocol (PPP). Link level bandwidth management is essentially the process of keeping track of all traffic and deciding whether an additional dial line or ISDN channel should be opened or an extraneous one closed. The field of this invention is concerned with network level bandwidth management, i.e. policies to assign available bandwidth from a single logical link to network flows.

In a copending U.S. patent application Ser. No. 08/742,994, now U.S. Pat. No. 6,038,216, in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Environment Without Data Rate Supervision," a technique for automatically scheduling TCP packets for transmission is disclosed. Furthermore, in U.S. Pat. No. 5,802,106, in the name of Robert L. Packer, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision," a technique for automatically determining the data rate of a TCP connection is disclosed. Finally, in a copending U.S. Pat. application Ser. No. 08/977,376, now abandoned, in the name of Robert L. Packer, entitled "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers in Store and Forward Network," a technique for automatically allocating bandwidth based upon data rates of TCP connections according to a hierarchical classification paradigm is disclosed.

Automated tools assist the network manager in configuring and managing the network equipped with the rate control techniques described in these copending applications. In a related copending application, a tool is described which enables a network manager to automatically produce policies for traffic being automatically detected in a network. It is described in a copending U.S. patent application Ser. No. 09/198,051, still pending, in the name of Guy Riddle, entitled "Method for Automatically Determining a Traffic Policy in a Packet Communications Network", based on U.S. Provisional Patent Application Ser. No. 60/066,864. The subject of the present invention is also a tool designed to assist the network manager.

While these efforts teach methods for solving problems associated with scheduling transmissions, automatically determining data flow rate on a TCP connection, allocating bandwidth based upon a classification of network traffic and automatically determining a policy, respectively, there is no teaching in the prior art of methods for automatically classifying packet traffic based upon information gathered from a multiple layers in a multi-layer protocol network.

Bandwidth has become the expensive commodity of the '90s, as traffic expands faster than resources, the need to "prioritize" a scarce resource, becomes ever more critical. One way to solve this is by applying "policies" to control traffic classified as to type of service required in order to more efficiently match resources with traffic.

Traffic may be classified by type, e.g. E-mail, web surfing, file transfer, at various levels. For example, to classify by network paradigm, examining messages for an IEEE source/destination service access point (SAP) or a sub-layer access protocol (SNAP) yields a very broad indicator, i.e., SNA or IP. More specific types exist, such as whether an IP protocol field in an IP header indicates TCP or UDP. Well known connection ports provide indications at the application layer, i.e., SMTP or HTTP.

Classification is not new. Firewall products like "CheckPoint FireWall-1," a product of CheckPoint Software Technologies, Inc., a company with headquarters in Redwood City, Calif., have rules for matching traffic. Bandwidth managers such as "Aponet," a product of Aponet, Inc., a company with headquarters in San Jose, Calif., classify by destination. The PacketShaper, a product of Packeteer, Inc., a company with headquarters in Cupertino, Calif., allows a user to manually enter rules to match various traffic types for statistical tracking, i.e., counting by transaction, byte count, rates, etc. However, manual rule entry requires a level of expertise that limits the appeal for such a system to network savvy customers. What is really needed is a method for analyzing real traffic in a customer's network and automatically producing a list of the "found traffic."

SUMMARY OF THE INVENTION

According to the invention, in a packet communication environment, a method is provided for automatically classifying packet flows for use in allocating bandwidth resources and the like by a rule of assignment of a service level. The method comprises applying individual instances of traffic classification paradigms to packet network flows based on selectable information obtained from a plurality of layers of a multi-layered communication protocol in order to define a characteristic class, then mapping the flow to the defined traffic class. It is useful to note that the automatic classification is sufficiently robust to classify a complete enumeration of the possible traffic.

In the present invention network managers need not know the technical aspects of each kind of traffic in order to configure traffic classes and service aggregates bundle traffic to provide a convenience to the user, by clarifying processing and enables the user to obtain group counts of all parts comprising a service.

The invention will be better understood upon reference to the following detailed description in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A depicts a representative client server relationship in accordance with a particular embodiment of the invention;

FIG. 1B depicts a functional perspective of the representative client server relationship in accordance with a particular embodiment of the invention;

FIG. 1C depicts a representative internetworking environment in accordance with a particular embodiment of the invention;

FIG. 1D depicts a relationship diagram of the layers of the TCP/IP protocol suite;

FIGS. 2A–2B depict representative divisions of bandwidth;

FIG. 3 depicts a component diagram of processes and data structures in accordance with a particular embodiment of the invention; and

FIGS. 4A–4B depict flowcharts of process steps in automatically classifying traffic in accordance with a particular embodiment of the invention.

DESCRIPTION OF SPECIFIC EMBODIMENTS

1.0 Introduction

The present invention provides techniques to automatically classify a plurality of heterogeneous packets in a packet telecommunications system for management of network bandwidth in systems such as a private area network, a wide area network or an internetwork. Systems according to the present invention enable network managers to: automatically define traffic classes, for which policies may then be created for specifying service levels for the traffic classes and isolating bandwidth resources associated with certain traffic classes. Inbound as well as outbound traffic may be managed. Table 1 provides a definitional list of terminology used herein.

TABLE 1

LIST OF DEFINITIONAL TERMS

ADMISSIONS CONTROL	A policy invoked whenever a system according to the invention detects that a guaranteed information rate cannot be maintained. An admissions control policy is analogous to a busy signal in the telephone world.
CLASS SEARCH ORDER	A search method based upon traversal of a N-ary tree data structure containing classes.
COMMITTED INFORMATION RATE(CIR)	A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for a committed bandwidth. Also called a guaranteed information rate (GIR).
EXCEPTION	A class of traffic provided by the user which supersedes an automatically determined classification order.
EXCESS INFORMATION RATE(EIR)	A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for uncommitted bandwidth resources.
FLOW	A flow is a single instance of a traffic class. For example, all packets in a TCP connection belong to the same flow. As do all packets in a UDP session.
GUARANTEED INFORMATION RATE (GIR)	A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for a committed bandwidth. Also called a committed information rate (CIR).
INSIDE	On the system side of an access link. Outside clients and servers are on the other side of the access link.
ISOLATION	Isolation is the degree that bandwidth resources are allocable to traffic classes.
OUTSIDE	On the opposite side of an access link as viewed from the perspective of the system on which the software resides.
PARTITION	Partition is an arbitrary unit of network resources.
POLICY	A rule for the assignment of a service level to a flow.
POLICY INHERITANCE	A method for assigning policies to flows for which no policy exists in a hierarchical arrangement of policies. For example, if a flow is determined to be comprised of FTP packets for Host A, and no corresponding policy exists, a policy associated with a parent node, such as an FTP policy, may be located and used.
POLICY BASED SCALING	An adjustment of a requested data rate for a particular flow based upon the policy associated with the flow and information about the flow's potential rate.
SCALED RATE	Assignment of a data rate based upon detected speed.
SERVICE LEVEL	A service paradigm having a combination of characteristics defined by a network manager to handle a particular class of traffic. Service levels may be designated as either reserved or unreserved.
TRAFFIC CLASS	All traffic between a client and a server endpoints. A single instance of a traffic class is called a flow. Traffic classes have properties or class attributes such as, directionality, which is the property of traffic to be flowing inbound or outbound;
UNRESERVED SERVICE	Unreserved service is a service level defined in terms of priority in which no reservation of bandwidth is made.
URI	A Universal Resource Identifier is the name of the location field in a web reference address. It is also called a URL or Universal Resource Locator

1.1 Hardware Overview

The method for automatically classifying heterogeneous packets in a packet telecommunications environment of the present invention is implemented in the C programming language and is operational on a computer system such as shown in FIG. **1A**. This invention may be implemented in a client-server environment, but a client-server environment is not essential. This figure shows a conventional client-server computer system which includes a server **20** and numerous clients, one of which is shown as client **25**. The use of the term "server" is used in the context of the invention, wherein the server receives queries from (typically remote) clients, does substantially all the processing necessary to formulate responses to the queries, and provides these responses to the clients. However, server **20** may itself act in the capacity of a client when it accesses remote databases located at another node acting as a database server.

The hardware configurations are in general standard and will be described only briefly. In accordance with known practice, server **20** includes one or more processors **30** which communicate with a number of peripheral devices via a bus subsystem **32**. These peripheral devices typically include a storage subsystem **35**, comprised of a memory subsystem **35***a* and a file storage subsystem **35***b* holding computer programs (e.g., code or instructions) and data, a set of user interface input and output devices **37**, and an interface to outside networks, which may employ Ethernet, Token Ring, ATM, IEEE 802.3, ITU X.25, Serial Link Internet Protocol (SLIP) or the public switched telephone network. This interface is shown schematically as a "Network Interface" block **40**. It is coupled to corresponding interface devices in client computers via a network connection **45**.

Client **25** has the same general configuration, although typically with less storage and processing capability. Thus, while the client computer could be a terminal or a low-end personal computer, the server computer is generally a high-end workstation or mainframe, such as a SUN SPARC server. Corresponding elements and subsystems in the client computer are shown with corresponding, but primed, reference numerals.

Bus subsystem **32** is shown schematically as a single bus, but a typical system has a number of buses such as a local bus and one or more expansion buses (e.g., ADB, SCSI, ISA, EISA, MCA, NuBus, or PCI), as well as serial and parallel ports. Network connections are usually established through a device such as a network adapter on one of these expansion buses or a modem on a serial port. The client computer may be a desktop system or a portable system.

The user interacts with the system using interface devices **37'** (or devices **37** in a standalone system). For example, client queries are entered via a keyboard, communicated to client processor **30'**, and thence to modem or network interface **40'** over bus subsystem **32'**. The query is then communicated to server **20** via network connection **45**. Similarly, results of the query are communicated from the server to the client via network connection **45** for output on one of devices **37'** (say a display or a printer), or may be stored on storage subsystem **35'**.

FIG. **1B** is a functional diagram of a computer system such as that of FIG. **1A**. FIG. **1B** depicts a server **20**, and a representative client **25** of a plurality of clients which may interact with the server **20** via the Internet **45** or any other communications method. Blocks to the right of the server are indicative of the processing steps and functions which occur in the server's program and data storage indicated by blocks **35***a* and **35***b* in FIG. **1A**. A TCP/IP "stack" **44** works in conjunction with Operating System **42** to communicate with processes over a network or serial connection attaching Server **20** to Internet **45**. Web server software **46** executes concurrently and cooperatively with other processes in server **20** to make data objects **50** and **51** available to requesting clients. A Common Gateway Interface (CGI) script **55** enables information from user clients to be acted upon by web server **46**, or other processes within server **20**. Responses to client queries may be returned to the clients in the form of a Hypertext Markup Language (HTML) document outputs which are then communicated via Internet **45** back to the user.

Client **25** in FIG. **1B** possesses software implementing functional processes operatively disposed in its program and data storage as indicated by block **35***a'* in FIG. **1A**. TCP/IP stack **44'**, works in conjunction with Operating System **42'** to communicate with processes over a network or serial con-

nection attaching Client **25** to Internet **45**. Software implementing the function of a web browser **46**'executes concurrently and cooperatively with other processes in client **25** to make requests of server **20** for data objects **50** and **51**. The user of the client may interact via the web browser **46**' to make such queries of the server **20** via Internet **45** and to view responses from the server **20** via Internet **45** on the web browser **46**'.

Network Overview

FIG. 1C is illustrative of the internetworking of a plurality of clients such as client **25** of FIGS. **1A** and **1B** and a plurality of servers such as server **20** of FIGS. **1A** and **1B** as described herein above. In FIG. **1C**, network **60** is an example of a Token Ring or frame oriented network. Network **60** links host **61**, such as an IBM RS6000 RISC workstation, which may be running the AIX operating system, to host **62**, which is a personal computer, which may be running Windows **95**, IBM OS/2 or a DOS operating system, and host **63**, which may be an IBM AS/400 computer, which may be running the OS/400 operating system. Network **60** is internetworked to network **70** via a system gateway which is depicted here as router **75**, but which may also be a gateway having a firewall or a network bridge. Network **70** is an example of an Ethernet network that interconnects host **71**, which is a SPARC workstation, which may be running SUNOS operating system with host **72**, which may be a Digital Equipment VAX6000 computer which may be running the VMS operating system.

Router **75** is a network access point (NAP) of network **70** and network **60**. Router **75** employs a Token Ring adapter and Ethernet adapter. This enables router **75** to interface with the two heterogeneous networks. Router **75** is also aware of the Inter-network Protocols, such as ICMP and RIP, which are described herein below.

FIG. 1D is illustrative of the constituents of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. The base layer of the TCP/IP protocol suite is the physical layer **80**, which defines the mechanical, electrical, functional and procedural standards for the physical transmission of data over communications media, such as, for example, the network connection **45** of FIG. **1A**. The physical layer may comprise electrical, mechanical or functional standards such as whether a network is packet switching or frame-switching; or whether a network is based on a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) or a frame relay paradigm.

Overlying the physical layer is the data link layer **82**. The data link layer provides the function and protocols to transfer data between network resources and to detect errors that may occur at the physical layer. Operating modes at the datalink layer comprise such standardized network topologies as IEEE 802.3 Ethernet, IEEE 802.5 Token Ring, ITU X.25, or serial (SLIP) protocols.

Network layer protocols **84** overlay the datalink layer and provide the means for establishing connections between networks. The standards of network layer protocols provide operational control procedures for internetworking communications and routing information through multiple heterogenous networks. Examples of network layer protocols are the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP). The Address Resolution Protocol (ARP) is used to correlate an Internet address and a Media Access Address (MAC) for a particular host. The Routing Information Protocol (RIP) is a dynamic routing protocol for passing routing information between hosts on networks. The Internet Control Message Protocol (ICMP) is an internal protocol for passing control messages between hosts on various networks. ICMP messages provide feedback about events in the network environment or can help determine if a path exists to a particular host in the network environment. The latter is called a "Ping". The Internet Protocol (IP) provides the basic mechanism for routing packets of information in the Internet. IP is a non-reliable communication protocol. It provides a "best efforts" delivery service and does not commit network resources to a particular transaction, nor does it perform retransmissions or give acknowledgments.

The transport layer protocols **86** provide end-to-end transport services across multiple heterogenous networks. The User Datagram Protocol (UDP) provides a connectionless, datagram oriented service which provides a non-reliable delivery mechanism for streams of information. The Transmission Control Protocol (TCP) provides a reliable session-based service for delivery of sequenced packets of information across the Internet. TCP provides a connection oriented reliable mechanism for information delivery.

The session, or application layer **88** provides a list of network applications and utilities, a few of which are illustrated here. For example, File Transfer Protocol (FTP) is a standard TCP/IP protocol for transferring files from one machine to another. FTP clients establish sessions through TCP connections with FTP servers in order to obtain files. Telnet is a standard TCP/IP protocol for remote terminal connection. A Telnet client acts as a terminal emulator and establishes a connection using TCP as the transport mechanism with a Telnet server. The Simple Network Management Protocol (SNMP) is a standard for managing TCP/IP networks. SNMP tasks, called "agents", monitor network status parameters and transmit these status parameters to SNMP tasks called "managers." Managers track the status of associated networks. A Remote Procedure Call (RPC) is a programming interface which enables programs to invoke remote functions on server machines. The Hypertext Transfer Protocol (HTTP) facilitates the transfer of data objects across networks via a system of uniform resource indicators (URI).

The Hypertext Transfer Protocol is a simple protocol built on top of Transmission Control Protocol (TCP). It is the mechanism which underlies the function of the World Wide Web. The HTTP provides a method for users to obtain data objects from various hosts acting as servers on the Internet.

2.0 Traffic Class

A traffic class is broadly defined as traffic between one or more clients and one or more servers. A single instance of a traffic class is called a flow. Traffic classes have the property, or class attribute, of being directional, i.e. all traffic flowing inbound will belong to different traffic classes and be managed separately from traffic flowing outbound. The directional property enables asymmetric classification and control of traffic, i.e., inbound and outbound flows belong to different classes which may be managed independent of one another.

Traffic classes may be defined at any level of the IP protocol as well as for other non-IP protocols. For example, at the IP level, traffic may be defined as only those flows between a specificed set of inside and outside IP addresses or domain names. An example of such a low level traffic class definition would be all traffic between my network and other corporate offices throughout the Internet. At the application level, traffic classes may be defined for specific URIs within a web server. Traffic classes may be defined having "Web aware" class attributes. For example, a traffic class

could be created such as all URIs matching "*.html" for all servers, or all URI patterns matching "*.gif' for server X, or for access to server Y with URI pattern "/sales/*" from client Z, wherein '*' is a wildcard character, i.e., a character which matches all other character combinations. Traffic class attributes left unspecified will simply match any value for that attribute. For example, a traffic class that accesses data objects within a certain directory path of a web server is specified by a URI pattern of the directory path to be managed, e.g. "/sales/*" .

2.1 Classifying Traffic

The present invention provides a method for classifying traffic according to a definable set of classification attributes selectable by the manager, including selecting a subset of traffic of interest to be classified. The invention provides the ability to classify and search traffic based upon multiple orthogonal classification attributes.

Traffic class membership may be hierarchical. Thus, a flow may be classified by a series of steps through a traffic class tree, with the last step (i.e., at the leaves on the classification tree) mapping the flow to a policy. The policy is a rule of assignment for flows. Web traffic may also be classified by HTTP header types such as Content-Type (MIME type) or User-Agent.

A classification tree is a data structure representing the hierarchical aspect of traffic class relationships. Each node of the classification tree represents a class, and has a traffic specification, i.e., a set of attributes or characteristics describing the traffic associated with it. Leaf nodes of the classification tree may contain policies. According to a particular embodiment, the classification process checks at each level if the flow being classified matches the attributes of a given traffic class. If it does, processing continues down to the links associated with that node in the tree. If it does not, the class at the level that matches determines the policy for the flow being classified. If no policy specific match is found, the flow is assigned the default policy.

In a preferred embodiment, the classification tree is an N-ary tree with its nodes ordered by specificity. For example, in classifying a particular flow in a classification tree ordered first by organizational departments, the attributes of the flow are compared with the traffic specification in each successive department node and if no match is found, then processing proceeds to the next subsequent department node. If no match is found, then the final compare is a default "match all" category. If, however, a match is found, then classification moves to the children of this department node. The child nodes may be ordered by an orthogonal paradigm such as, for example, "service type." Matching proceeds according to the order of specificity in the child nodes. Processing proceeds in this manner, traversing downward and from left to right in FIGS. 2A and 2B, which describe a classification tree, searching the plurality of orthogonal paradigms. Key to implementing this a hierarchy is that the nodes are arranged in decreasing order of specificity. This permits search to find the most specific class for the traffic before more general.

Table 2 depicts components from which Traffic classes may be built. Note that the orientation of the server (inside or outside) is specified. And as noted above, any traffic class component may be unspecified, i.e. set to match any value.

TABLE 2

Components of a Traffic Class Specifier

Inside (Client or Server)	Global	Outside (Server or Client)
IP Address/Domain Name	TCP or UDP Service	IP Address/Domain Name
Port Number	e.g., WWW,	Port Number
MAC Address	FTP, RealAudio, etc.	MAC Address
	URI pattern for Web Service,	
	MIME type for Web Service	
	IPX Service	
	SNA Service	
	LAT Service	
	IP precedence	

FIGS. 2A and 2B depict representative allocations of bandwidth made by a hypothetical network manager as an example. In FIG. 2A, the network manager has decided to divide her network resources first by allocating bandwidth between Departments A and B. FIG. 2A shows the resulting classification tree **201**, in which Department A bandwidth resources **202** and Department B bandwidth resources **204** each have their own nodes representing a specific traffic class for that department. Each traffic class may have a policy attribute associated with it. For example, in FIG. 2A, the Department A resources node **202** has the policy attribute Inside Host Subnet A associated with it. Next, the network manager has chosen to divide the bandwidth resources of Department A among two applications. She allocates an FTP traffic class **206** and a World Wide Web server traffic class **208**. Each of these nodes may have a separate policy attribute associated with them. For example, in FIG. 2A, the FTP node **206** for has an attribute Outside port **20** associated with it. Similarly, the network manager has chosen to divide network bandwidth resources of Department B into an FTP server traffic class **210** and a World Wide Web server traffic class **212**. Each may have their own respective policies.

FIG. 2B shows a second example **203**, wherein the network manager has chosen to first divide network bandwidth resource between web traffic and TCP traffic. She creates three traffic nodes, a web traffic node **220**, a TCP traffic node **224** and a default node **225**. Next, she divides the web traffic among two organizational departments by creating a Department A node **226**, and a Department B node **228**. Each may have its own associated policy. Similarly, she divides TCP network bandwidth into separate traffic classes by creating a Department A node **230** and a Department B node **232**. Each represents a separate traffic class which may have its own policy.

All traffic which does not match any user specified traffic class falls into an automatically created default traffic class which has a default policy. In FIG. 2A, the default category is depicted by a default node **205**, and in FIG. 2B, the default category is depicted by a default node **225**.

3.0 Automatically Classifying Traffic

3.1 Automatic Traffic Classification

Network traffic is automatically classified under existing classes, beginning with the broadest classes, an inbound traffic class and an outbound traffic class, in protocol layer independent categories. For example, a particular instance of traffic may be classified according to its transport layer characteristics, e.g., Internet Protocol port number, as well as its application layer information, e.g., SMTP. Characteristics such as MIME types may also be automatically identified. Standard protocols, such as, IPX, SNA, and

services, such as, SMTP and FTP are recognized for automatic classification. Classification is performed to the most specific level determinable. For example, in select embodiments, non-IP traffic, such as SNA, may be classified only by protocol, whereas Internet Protocol traffic may be classified to the /etc/services level. Classification beyond a terminal classification level is detected and prevented. For example, in a select embodiment, a class matching "ipx" or "nntp" will not be further automatically classified.

3.1.1 Service Aggregates

A service aggregate is provided for certain applications that use more than one connection in a particular conversation between a client and a server. For example, an FTP client in conversation with an FTP server employs a command channel and a transfer channel, which are distinct TCP sessions on two different ports. In cases where two or three TCP or UDP sessions exist for each conversation between one client and one server, it is useful to provide a common traffic class i.e., the service aggregate, containing the separate conversations. In practice, these types of conversations are between the same two hosts, but use different ports. According to the invention, a class is created with a plurality of traffic specifications, each matching various component conversations.

3.1.2 Subclassification Under Specified Criterion

Subclassification of traffic into a tree is performed by matching the hosts and then searching for particular services. Traffic specifications are aggregate kinds of traffic for a traffic class, e.g., different components of FTP may reside under class FTP. Subclassification is performed by first locating a class that matches, and then performing finer grade matchings. Processing commences with a decision on what traffic is to be subclassified. A marker is placed in the match_all default node so that when match processing reaches the marker, the autoclassification processing depicted in flowchart **403**, determines that it has not found an existing class for the traffic being classified.

3.1.3 Default Suggested Policies

A default policy may be suggested or, in select embodiments, automatically applied, to a traffic class which has been automatically classified. Applying suggested or default policies for a new class at a user's option is described in a copending, commonly owned, U.S. patent application Ser. No. 09/198,051, still pending, entitled, "Method for Automatically Determining a Traffic Policy in a Packet Communications Network", which is incorporated herein by reference in its entirety for all purposes.

3.1.4 Analysis of Data in Determining Traffic Class

In a preferable embodiment, classification can extend to examination of the data contained in a flow's packets. Certain traffic may be distinguished by a signature even if it originates with a server run on a non-standard port, for example, an HTTP conversation on port **8080** would not be otherwise determinable as HTTP from the port number. Further analysis of the data is conducted in order to determine classification in instances where: 1) FTP commands are used to define server ports, 2) HTTP protocol is used for non-web purposes. The data is examined for indication of push traffic, such as pointcast, which uses HTTP as a transport mechanism. These uses may be isolated and classified into a separate class. Marimba and pointcast can be distinguished by looking into the data for a signature content header in the get request. Pointcast has URLs that begin with "/FIDO-1/." Other applications in which protocol can be inferred from data include Telnet traffic. Both tn3270 and tn3270E (emulation) may be detected by looking into data and given a different class. Telnet traffic has option negotiations which may indicate an appropriate class.

3.1.5 Identity of Traffic Based Upon Resource Creator's Class

A traffic class may be inferred from determining the identity of the creator of a resource used by the traffic class. For example, the identity of traffic using a certain connection can be determined by finding the identity of the creator of the connection. This method is used to detect Real Time Protocol (RTP) for point-to-point telephony, RTP for broadcast streaming, CCITT/ITU H320-telephony over ISDN, H323-internet telephony over the internet (bidirectional) and RTSP real time streaming protocol for movies (unidirectional).

3.1.6 Dynamic Ports

Applications having a well known port for a server may make use of dynamic ports. Some applications will send initial messages across a first connection, then negotiate a dynamic port for further conversation. During the existence of a connection, both endpoints are known. A check is made for two simultaneous connections to the same, non well-known port, at same time from different locations. This condition is indicative of a connection port for some application. Varieties of the dynamic port exist in applications. Certain dynamic ports are incorporated into a client. Others are fixed but not registered. Still others are negotiated during a protocol exchange, as for example in passive FTP.

3.2 Automatic Traffic Classification Processing

FIG. 3 depicts components of a system for automatically classifying traffic according to the invention. A traffic tree **302** in which new traffic will be classified under a particular member class node. A traffic classifier **304** detects services for incoming traffic. Alternatively, the classifier may start with a service and determine the hosts using it. A knowledge base **306** contains heuristics for determining traffic classes. The knowledge base may be embodied in a file or a relational database. In a particular embodiment, the knowledge is contained within a data structure resident in memory. A plurality of saved lists **308** stores classified traffic pending incorporation into traffic tree **302**. In select embodiments, entries for each instance of traffic may be kept. In alternate embodiments, a copy of an entry and a count of duplicate copies for the entry is maintained.

FIG. 4A depicts a flowchart **401** of processing steps for automatically classifying traffic. In a step **402**, a flow specification is parsed from the flow being classified. Then in a step **404**, the flow specification parsed from the flow in step **402** is compared with the traffic specifications in each node of the classification tree. Rules are checked starting from most specific to least specific. In a decisional step **406**, a determination is made if traffic matches one of the classes being classified. If this is so, then in a step **408**, an entry is made in a list of identifying characteristics, such as protocol type (SAP), IP protocol number, server port, traffic type if known, MIME type, a time of occurrence of the traffic. In an optional step **410**, duplicate instances having the same identifying characteristics are suppressed, in favor of keeping a count of the duplicates and a most recent time traffic with these identifying characteristics was encountered. In an optional step **412**, a byte count of traffic of this type has been detected is included. Otherwise, the automatic classification has failed to determine a class and processing returns.

In a preferable embodiment, processing according to flowchart **401** may execute on multiple instances of saved list **308**.

3.2.1 Displaying Results to a User

In an optional step **413** (not show), after the processing of flowchart **401** completes or at periodic intervals or on demand, a list of traffic classes produced in steps **402**

through 412 are displayed to a network manager. The list may be sorted by any well-known criteria such as: 1) most "hits" during a recent interval, 2) most recently-seen (most recent time first), 3) most data transferred (bytes/second) during some interval, or a moving average. The user may choose an interval length or display cutoff point (how many items, how recent, at least B bytes per second, or other thresholds). The Network manager may then take some action (e.g. pushing a button) to select the traffic types she wishes to add to the classification tree. The display can be hierarchical, as depicted in lines (3) below: FTP (3)

```
FTP-cmd
  FTP-data
to host1
  tcp
    FTP
      FTP-cmd
      FTP-data
    HTTP
      images
      java
      text
    port 9999
```

(3)

wherein the "port 9999" entry is an inference corresponding to an application checking for repeated or simultaneous connections made to a specific port.

In a related embodiment, a threshold for display or class creation of well-known traffic types is provided.

3.2.2 Interval Based Incorporation

In an alternative embodiment, at select intervals of time, non matching traffic is analyzed, and either 1) recognized and add to the tree, or 2) for repeated attempts to request a server connection port that is not known, upon exceeding a certain threshold, a class for the port's traffic is created and added to the classification tree.

FIG. 4B depicts a flowchart 403 of the processing steps for integrating traffic classes into a classification tree in an alternative embodiment. Processing steps of flowchart 403 periodically at a defined interval of seconds, having a value of 30 in the preferable embodiment, incorporate newly classified traffic into the classification tree. In a step 420, an instance of saved traffic is retrieved from the saved traffic list 308. Next in a decisional step 422, the instance of saved traffic is examined to determine whether it is well-known (e.g. registered SAP, protocol type, assigned port number) and a name representing its type exists. If this is so then processing continues with a test of whether the saved traffic belongs to a service aggregate in step 426. Otherwise, in a step 423 the instance of saved traffic is examined to determine whether it appears to be a server connection port of an unregistered IP port (or a port that has not been configured). If this is not so then, processing continues with the next traffic class in the saved list in step 420. In decisional step 426, the instance of saved traffic is examined to determine whether it belongs to a service aggregate. For example, an FTP session has one flow that is used to exchange commands and responses and a second flow that is used to transport data files. If the traffic does belong to a service aggregate, then in a step 428, a traffic class is created which will match all components of the service aggregate. In a further step 425, a new traffic class is created to match the instance of saved traffic. The class may be flat or hierarchical.

In an optional step, a suggested policy is determined for the traffic class created in step 425. Next, in a decisional step 432, a limit is checked to verify that the number of automatically created classes has not exceeded a specified maximum.

In a related embodiment in place of step 425, a display of traffic classes, sorted by most recently used, most hits, number of bytes received during any interval, which is determined by a plurality of time stamps, is available on demand to a network manager. The network manager indicates that the traffic is to be added to the tree.

In a particular embodiment a threshold is employed to determine traffic for which a separate class should be added. A minimum usage threshold indicates whether a particular port has been used at least n times in the last s seconds. If traffic is well known, i.e., SMTP, it is added to a traffic class immediately, i.e., threshold is equal to one, otherwise, the threshold is set equal to an arbitrary value, for example, eleven uses with not more than one minute between any two uses. A new class for traffic is given a generic name, e.g., Port99 traffic. Entries for traffic over a certain maximum threshold, for example one minute old, is discarded.

In a related embodiment, another method of identifying an individual traffic class is to detect simultaneous connections to the same host port from different clients. This provides an indication that the port is a well-known connection port.

Traffic classes are created for any combination of the above mentioned categories. A flag is added to all traffic classes so created in order to indicate that it is the product of the auto classifier.

3.2 Command Language Interface:

In a particular embodiment, function of the classifier 304 is controlled by a command language interface. Table 3 depicts a plurality of command language interface commands.

TABLE 3

setup autoclassify {on|off}	To activate autoclassification for various classes to detect well-known protocols and services:
class auto <tclass> {inside|outside|both}	To turn on autoclassification under a class To detect services with the host on the inside, the outside, or both directions.
class auto <tclass> off	To turn off use

The new classes have names in the format of lines (4) below:

```
<direction>_<service>_<parent>or <direction>_port
  _<number>_<parent>or <direction>_<service>_
  <portnum>_<parent>
```

(4)

where <direction>is either "inside" or "outside" for TCP/UDP services or "auto" for others.

If a well-known service on a non-standard port (e.g. HTTP on 8080) is detected, a name in the last format will be created, assuming no previous class match.

If a heretofore unknown server-connection port appears to be "well used" , an entry of the second type is created. The threshold for creation is currently 11 hits with no more than 1 minute (granularity of checking is at least 30 seconds between running successive autoclassification processes) between any two hits. For example, see lines (5) below:

```
inbound/inside/ftp
outside http inbound
auto_sna_inbound
inside_port_505_luna
outside_pointcast_8888_inbound
```

(5)

The "class show" command will now show an 'D' in the flags for classes currently being autoclassified.

3.3 Syntax of Traffic Specifications:

Flow specifications and traffic specifications have an inside service field and an outside service field. Each will have values of SVC_UNKNOWN (0), SVC_CLIENT (1), or a number greater than 1, which is the service index, an index into the global table gServiceTable. If a type of service is known for a connection, the service field at a particular side will be set to SVC_CLIENT and the service field at the opposite side will be the index into gServiceTable. If a type of service is not known for the traffic, both inside service field and outside service field will be SVC_UNKNOWN. A person of reasonable skill in the art will appreciate that other embodiments for the table, such as representing the information contained therein as text strings or by any one of a plurality of possible encoding schemes, are realizable without departing from the present invention.

Therefore, a traffic specification can have "outside service:http" (or just "outside HTTP") which is different than "outside tcp:80". The first will match HTTP on any port while the second will match anything on port 80 (including pointcast and marimba).

Specifying a traffic specification tspec "service:<agg>" returns traffic specifications for various traffic belonging to the service. Specifying "class new inbound cdls outside dls" is the same as "class new inbound cdls outside service:dls-wpn" and "class tspec add cdls outside service:dls-rpn". Most auto-recognized services will create a class that encompasses all the pieces.

Network managers need not be aware of services which are known to be derivative of others, e.g., pointcast and marimba are special cases of HTTP and tn3270 is a special case of Telnet, in order to work with the system.

4.0 Conclusion

In conclusion, the present invention provides for an automatic determination of a policy for a packet telecommunications systems wherein bandwidth is allocated to requesting flows according to automatically determined application requirements. An advantage of traffic classification techniques according to the present invention is that network managers need not know the technical aspects of each kind of traffic in order to configure traffic classes. A further advantage of the present invention is that traffic classes may include information such as a MIME type for web traffic .

Other embodiments of the present invention and its individual components will become readily apparent to those skilled in the art from the foregoing detailed description. As will be realized, the invention is capable of other and different embodiments, and its several details are capable of modifications in various obvious respects, all without departing from the spirit and the scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not as restrictive. It is therefore not intended that the invention be limited except as indicated by the appended claims.

What is claimed is:

1. A method for automatically classifying traffic in a packet communications network, said network having any number of flows, including zero, comprising the steps of:

parsing a packet into a first flow specification, wherein said first flow specification contains at least one instance of any one of the following:

a protocol family designation,

a direction of packet flow designation,

a protocol type designation,

a pair of hosts,

a pair of ports,

in HTTP protocol packets, a pointer to a MIME type; thereupon,

matching the first flow specification of the parsing step to a plurality of classes represented by a plurality nodes of a classification tree type, each said classification tree type node having a traffic specification; thereupon,

if a matching classification tree type node was not found in the matching step, associating said first flow specification with one or more newly-created classification tree type nodes; thereupon,

incorporating said newly-created classification tree type nodes into said plurality of classification tree type nodes.

2. The method of claim 1 further comprising the steps of:

for at least a second flow having a second flow specification, recognizing said second flow specification and said first flow specification to comprise together a service aggregate; thereupon,

associating said first flow specification and said second flow specification with a newly-created classification tree node, said newly-created classification tree type node having a first traffic specification corresponding to said first flow specification and a second traffic specification corresponding to said second flow specification.

3. The method of claim 1 further comprising the steps of:

applying policies from said newly-created classification tree type nodes to instances of detected traffic.

4. The method of claim 1 further comprising the steps of:

for a subclassification under a specified criterion, if a matching classification tree type node was found in the matching step, said matching classification tree type node having at least one child classification tree type node, applying the matching, associating, and incorporating steps to a particular child classification tree type node of said matching classification tree type node as a part of classification.

5. The method of claim 1 wherein the parsing step further comprises the steps of:

examining data contained within a plurality of component packets belonging to said first flow for any number of a plurality of indicators of any of the following:

a protocol;

a service; thereupon, matching said plurality of indicators to said classes represented by a plurality of said classification tree type nodes.

6. The method of claim 1 further including measuring traffic load and invoking said classification upon achievement of a minimum usage threshold.

7. The method according to claim 1 wherein said matching step is applied to hierarchically-recognized classes.

8. A system for automatically classifying traffic in a packet telecommunications network, said network having any number of flows, including zero, comprising:

a plurality of network links upon which said traffic is carried;

a network routing means; and,

a processor means operative to:

parse a packet into a first flow specification, wherein said first flow specification contains at least one instance of any one of the following:

a protocol family designation,

a direction of packet flow designation,

a protocol type designation,

a pair of hosts,

a pair of ports,

in HTTP protocol packets, a pointer to a MIME type; thereupon,

match the first flow specification of the parsing step to a plurality of classes represented by a plurality of said classification tree type nodes, each said classification tree type node having a traffic specification and a mask, according to the mask; thereupon,

if a matching classification tree type node was not found in the matching step, associating said first flow specification with one or more newly-created classification tree type nodes; thereupon, incorporating said newly-created classification tree type nodes into said plurality of said classification tree type nodes.

9. The method of claim **8** further including measuring traffic load and invoking said classification upon achievement of a minimum usage threshold.

10. The method according to claim **8** wherein said matching step is applied to hierarchically-recognized classes.

11. A method for classifying traffic in a packet telecommunications network, said network having any number of flows, including zero, comprising the steps of:

parsing a packet into a first flow specification, wherein said first flow specification contains at least one instance of any one of the following:

a protocol family designation,

a direction of packet flow designation,

a protocol type designation,

a pair of hosts,

a pair of ports,

in HTTP protocol packets, a pointer to a MIME type; thereupon,

matching the first flow specification of the parsing step to a plurality of classes represented by a plurality of classification tree type nodes, each said classification tree type node having a traffic specification; thereupon,

if a matching classification tree type node was not found in the matching step, associating said first flow specification with at least one more newly-created node; thereupon,

displaying to a network administrator a representation of traffic according to said traffic specification for use in manual intervention.

12. The method according to claim **11** further including the step of sorting said traffic representation according to most recently occurring.

13. The method according to claim **11** further including the step of sorting said traffic representation according to most data transferred for a preselected period of time.

14. The method of claim **11** further including measuring traffic load and invoking said classification upon achievement of a minimum usage threshold.

15. The method according to claim **11** wherein said matching step is applied to hierarchically-recognized classes.

* * * * *

US006104700A

United States Patent [19]

Haddock et al.

[11] Patent Number: 6,104,700

[45] Date of Patent: Aug. 15, 2000

[54] **POLICY BASED QUALITY OF SERVICE**

[75] Inventors: **Stephen R. Haddock**, Los Gatos; **Justin N. Chueh**, Palo Alto; **Shehzad T. Merchant**, Mountain View; **Andrew H. Smith**, Palo Alto; **Michael Yip**, Sunnyvale, all of Calif.

[73] Assignee: **Extreme Networks**, Cupertino, Calif.

[21] Appl. No.: **09/018,103**

[22] Filed: **Feb. 3, 1998**

Related U.S. Application Data

[60] Provisional application No. 60/057,371, Aug. 29, 1997.

[51] **Int. Cl.**[7] **H04L 12/56**

[52] **U.S. Cl.** **370/235**; 370/412; 370/429

[58] **Field of Search** 370/412, 413, 370/414, 415, 416, 417, 418, 410, 229, 230, 231, 232, 233, 234, 235, 236, 237, 401, 402, 428, 429, 445

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,381,413	1/1995	Tobagi et al. .	
5,394,402	2/1995	Ross	370/94.1
5,499,238	3/1996	Shon	370/60.1
5,570,360	10/1996	Klausmeier et al.	370/412
5,629,937	5/1997	Hayter et al.	370/412
5,742,772	4/1998	Sreenan	395/200.56
5,748,614	5/1998	Wallmeier	370/412
5,796,719	8/1998	Peris et al.	370/418
5,850,399	12/1998	Ganmukhi et al.	370/412
5,859,835	1/1999	Varma et al.	370/412

OTHER PUBLICATIONS

International Search Report, PCT/US98/17817, 4 pages.

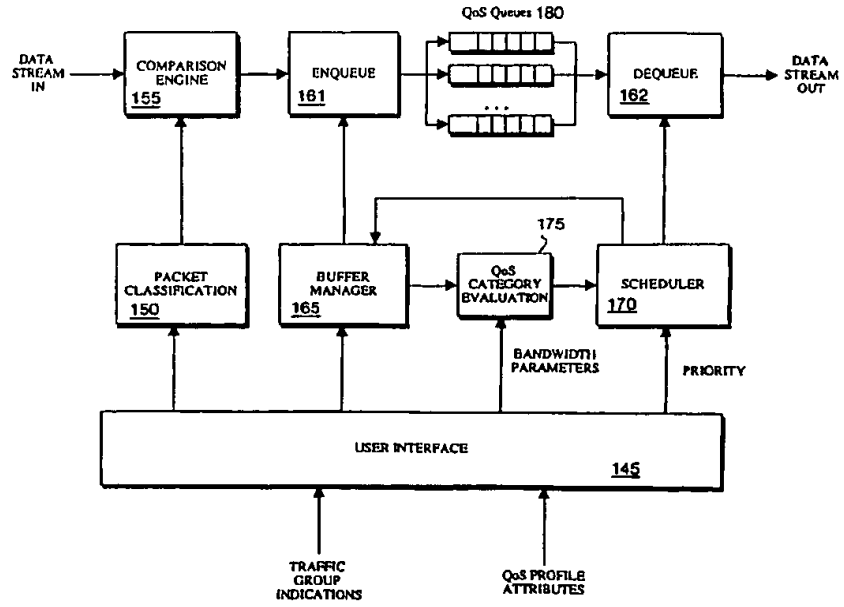
Primary Examiner—Huy D. Vu

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A flexible, policy-based, mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true quality of service (QoS) is provided. According to one aspect of the present invention, a method is provided for managing bandwidth allocation in a network that employs a non-deterministic access protocol, such as an Ethernet network. A packet forwarding device receives information indicative of a set of traffic groups, such as: a MAC address, or IEEE 802.1p priority indicator or 802.1Q frame tag, if the QoS policy is based upon individual station applications; or a physical port if the QoS policy is based purely upon topology. The packet forwarding device additionally receives bandwidth parameters corresponding to the traffic groups. After receiving a packet associated with one of the traffic groups on a first port, the packet forwarding device schedules the packet for transmission from a second port based upon bandwidth parameters corresponding to the traffic group with which the packet is associated. According to another aspect of the present invention, a method is provided for managing bandwidth allocation in a packet forwarding device. The packet forwarding device receives information indicative of a set of traffic groups. The packet forwarding device additionally receives information defining a QoS policy for the traffic groups. After a packet is received by the packet forwarding device, a traffic group with which the packet is associated is identified. Subsequently, rather than relying on an end-to-end signaling protocol for scheduling, the packet is scheduled for transmission based upon the QoS policy for the identified traffic group.

29 Claims, 5 Drawing Sheets



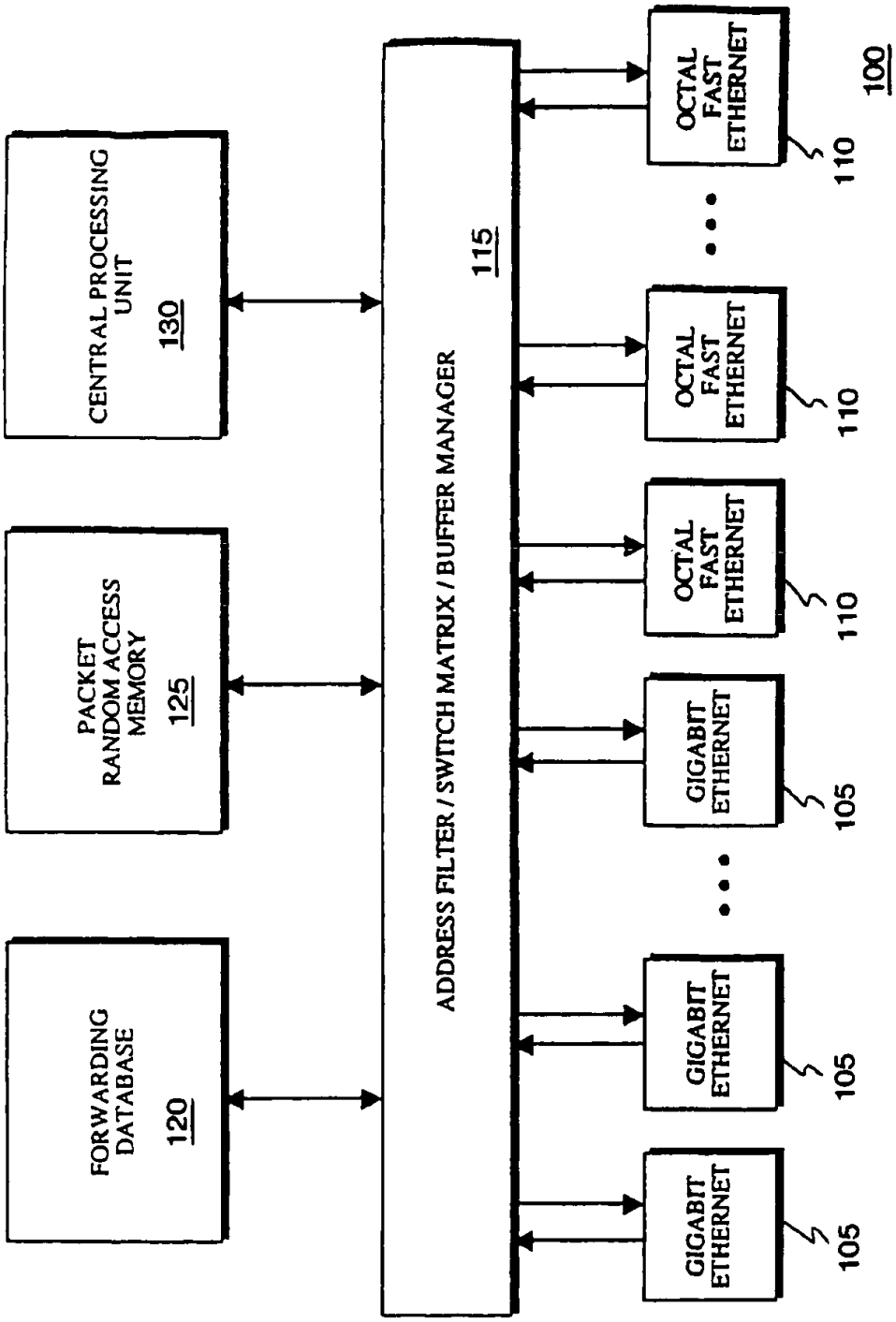

CENTRAL PROCESSING UNIT
130
PACKET RANDOM ACCESS MEMORY
125
FORWARDING DATABASE
120
ADDRESS FILTER / SWITCH MATRIX / BUFFER MANAGER
115
OCTAL FAST ETHERNET
110
GIGABIT ETHERNET
105
100

FIG. 1A

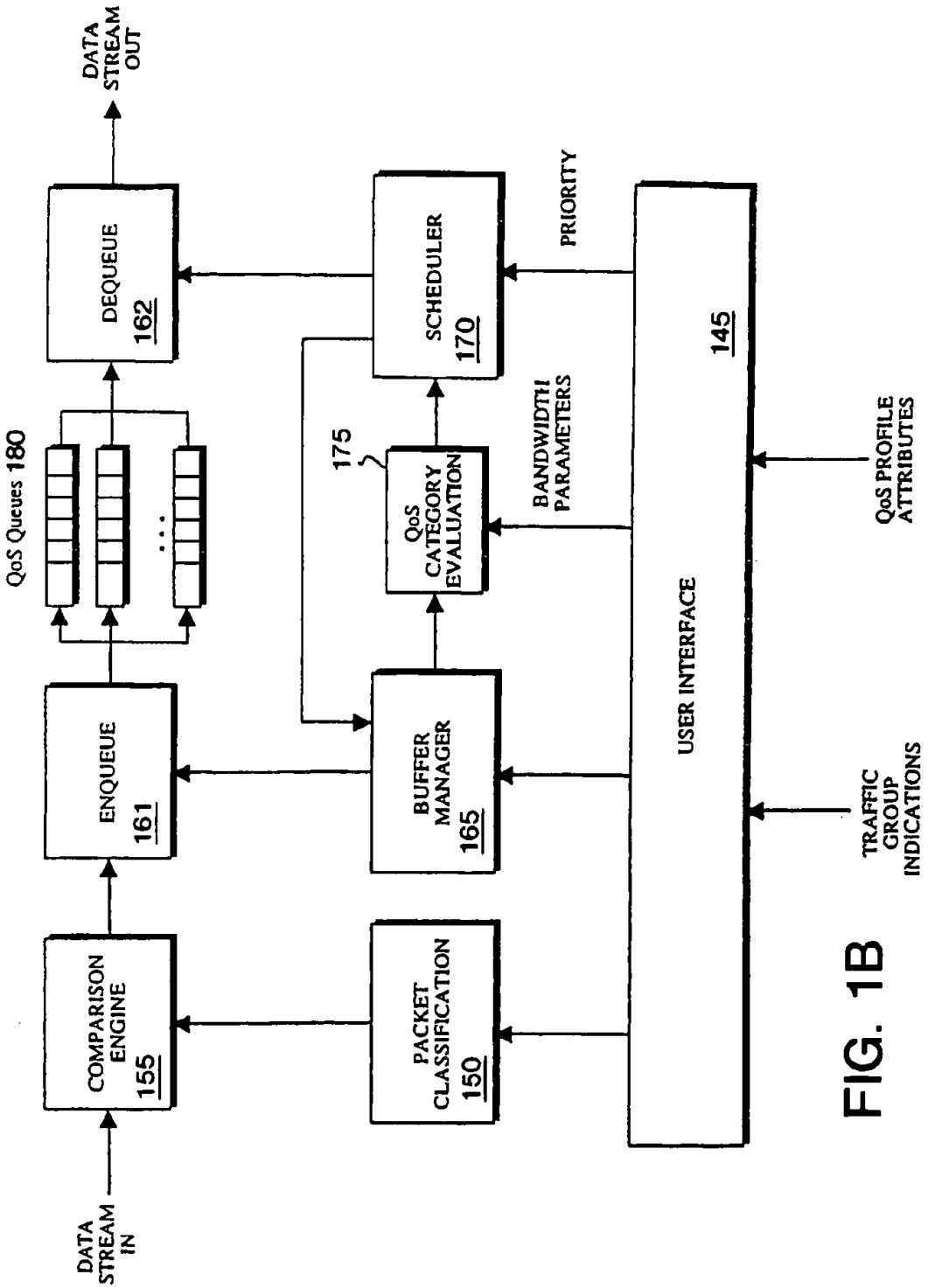

DATA STREAM IN
COMPARISON ENGINE 155
ENQUEUE 161
QoS Queues 180
DEQUEUE 162
DATA STREAM OUT
PACKET CLASSIFICATION 150
BUFFER MANAGER 165
175
QoS CATEGORY EVALUATION
SCHEDULER 170
PRIORITY
BANDWIDTH PARAMETERS
USER INTERFACE 145
QoS PROFILE ATTRIBUTES
TRAFFIC GROUP INDICATIONS

FIG. 1B

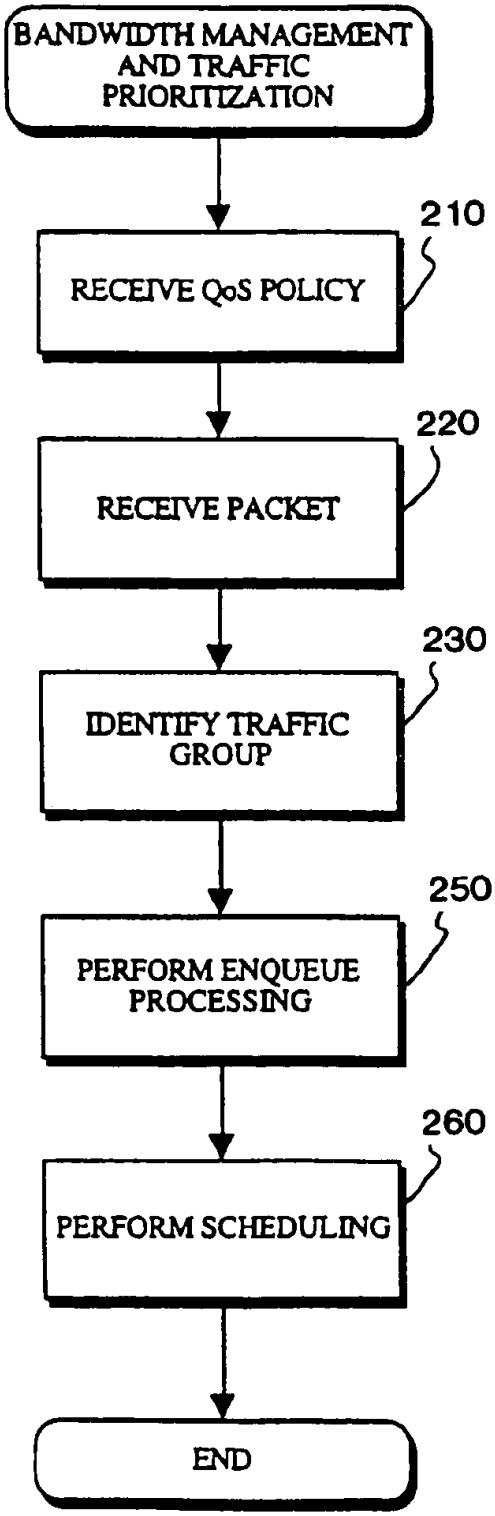

BANDWIDTH MANAGEMENT AND TRAFFIC PRIORITIZATION
210
RECEIVE QoS POLICY
220
RECEIVE PACKET
230
IDENTIFY TRAFFIC GROUP
250
PERFORM ENQUEUE PROCESSING
260
PERFORM SCHEDULING
END

FIG. 2

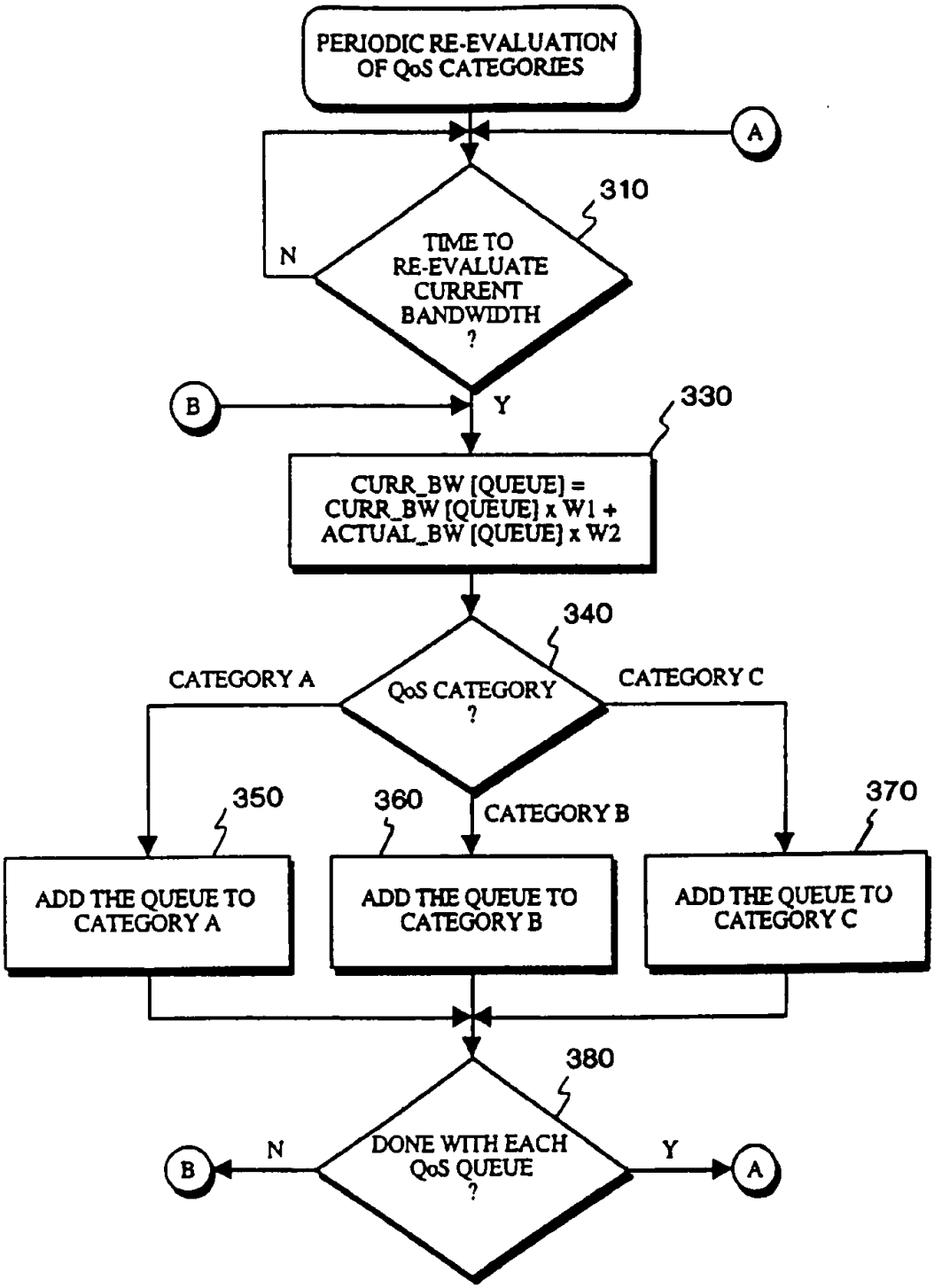

PERIODIC RE-EVALUATION OF QoS CATEGORIES
A
310
TIME TO RE-EVALUATE CURRENT BANDWIDTH ?
N
B
Y
330
CURR_BW [QUEUE] = CURR_BW [QUEUE] x W1 + ACTUAL_BW [QUEUE] x W2
340
QoS CATEGORY ?
CATEGORY A
CATEGORY B
CATEGORY C
350
360
370
ADD THE QUEUE TO CATEGORY A
ADD THE QUEUE TO CATEGORY B
ADD THE QUEUE TO CATEGORY C
380
DONE WITH EACH QoS QUEUE ?
N
Y
B
A

FIG. 3

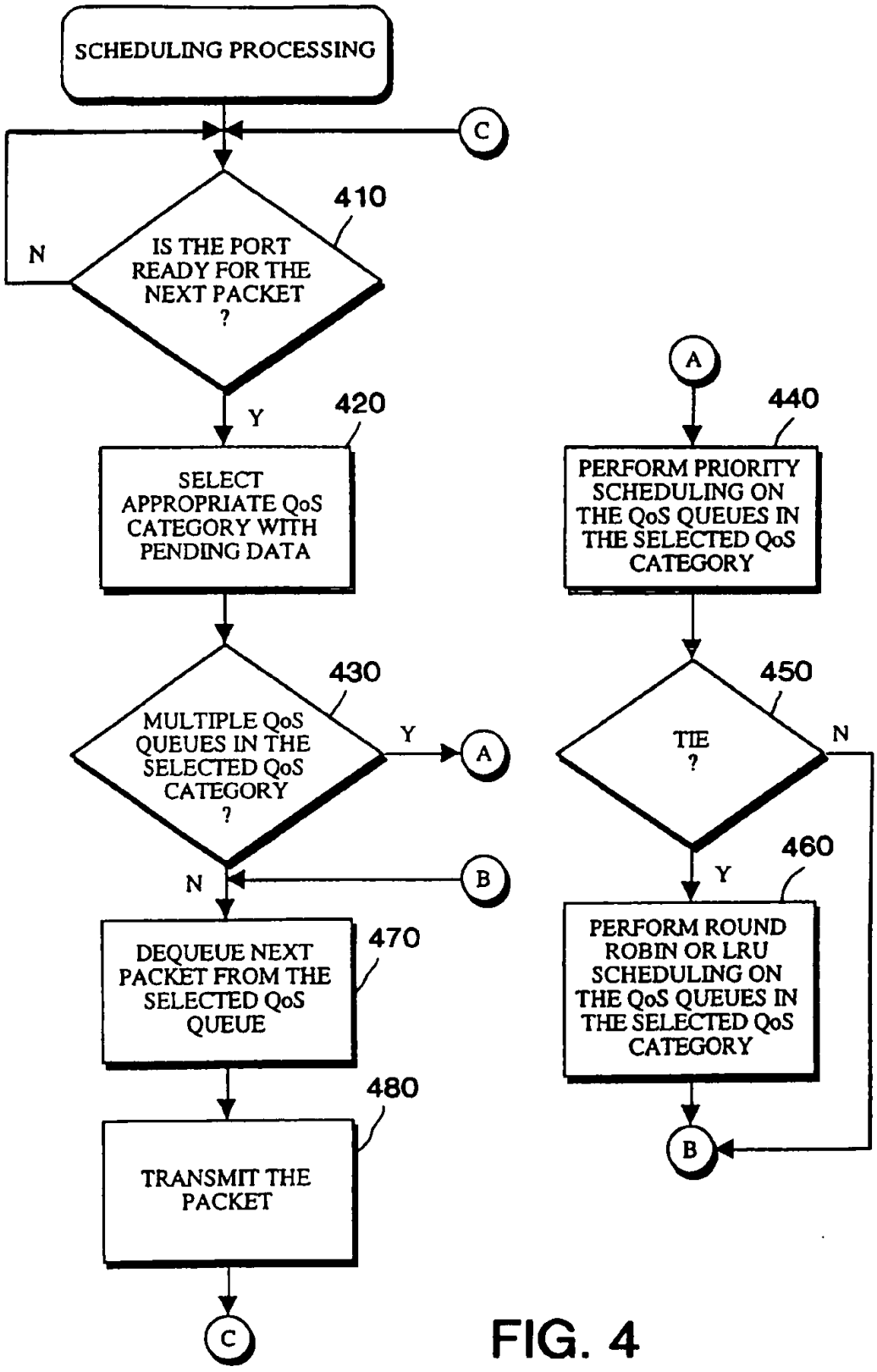

SCHEDULING PROCESSING
C
410
IS THE PORT READY FOR THE NEXT PACKET ?
N
Y
420
SELECT APPROPRIATE QoS CATEGORY WITH PENDING DATA
430
MULTIPLE QoS QUEUES IN THE SELECTED QoS CATEGORY ?
Y
A
N
B
470
DEQUEUE NEXT PACKET FROM THE SELECTED QoS QUEUE
480
TRANSMIT THE PACKET
C
A
440
PERFORM PRIORITY SCHEDULING ON THE QoS QUEUES IN THE SELECTED QoS CATEGORY
450
TIE ?
N
Y
460
PERFORM ROUND ROBIN OR LRU SCHEDULING ON THE QoS QUEUES IN THE SELECTED QoS CATEGORY
B

FIG. 4

POLICY BASED QUALITY OF SERVICE

This application claims the benefit of U.S. Provisional Application No. 60/057,371, filed Aug. 29, 1997.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of computer networking devices. More particularly, the invention relates to a flexible, policy-based mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true Quality of Service (QoS).

2. Description of the Related Art

Network traffic today is more diverse and bandwidth-intensive than ever before. Today's intranets are expected to support interactive multimedia, full-motion video, rich graphic images and digital photography. Expectations about the quality and timely presentation of information received from networks is higher than ever. Increased network speed and bandwidth alone will not satisfy the high demands of today's intranets.

The Internet Engineering Task Force (IETF) is working on a draft standard for the Resource Reservation Protocol (RSVP), an Internet Protocol-(IP) based protocol that allows end-stations, such as desktop computers, to request and reserve resources within and across networks. Essentially, RSVP is an end-to-end protocol that defines a means of communicating the desired Quality of Service between routers. RSVP is receiver initiated. The end-station that is receiving the data stream communicates its requirements to an adjacent router and those requirements are passed back to all intervening routers between the receiving end-station and the source of the data stream and finally to the source of the data stream itself. Therefore, it should be apparent that RSVP must be implemented across the whole network. That is, both end-stations (e.g., the source and destination of the data stream) and every router in between should be RSVP compliant in order to accommodate the receiving end-station's request.

While RSVP allows applications to obtain some degree of guaranteed performance, it is a first-come, first-served protocol, which means if there are no other controls within the network, an application using RSVP may reserve and consume resources that could be needed or more effectively utilized by some other mission-critical application. A further limitation of this approach to resource allocation is the fact that end-stations and routers must be altered to be RSVP compliant. Finally, RSVP lacks adequate policy mechanisms for allowing differentiation between various traffic flows. It should be appreciated that without a policy system in place, the network manager loses control.

Recent attempts to facilitate traffic differentiation and prioritization include draft standards specified by the Institute of Electrical and Electronics Engineers (IEEE). The IEEE 802.1Q draft standard provides a packet format for an application to specify which Virtual Local Area Network (VLAN) a packet belongs to and the priority of the packet. The IEEE 802.1p committee provides a guideline to classify traffic based on a priority indicator in an 802.1Q frame tag. This allows VLANs to be grouped into eight different traffic classes or priorities. The IEEE 802.1p committee does not, however, define the mechanism to service these traffic classes.

What is needed is a way to provide true Quality of Service ("QoS") in a network employing a non-deterministic access protocol, such as an Ethernet network, that not only has the ability to prioritize and service different traffic classes, but additionally provides bandwidth management and guarantees a quantifiable measure of service for packets associated with a particular traffic class. More specifically, with respect to bandwidth management, it is desirable to employ a weighted fair queuing delivery schedule which shares available bandwidth so that high priority traffic is usually sent first, but low priority traffic is still guaranteed an acceptable minimum bandwidth allocation. Also, it is desirable to centralize the control over bandwidth allocation and traffic priority to allow for QoS without having to upgrade or alter end-stations and existing routers as is typically required by end-to-end protocol solutions. Further, it would be advantageous to put the control in the hands of network managers by performing bandwidth allocation and traffic prioritization based upon a set of manager-defined administrative policies. Finally, since there are many levels of control a network manager may elect to administer, it is desirable to provide a variety of scheduling mechanisms based upon a core set of QoS profile attributes.

BRIEF SUMMARY OF THE INVENTION

A flexible, policy-based, mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true Quality of Service (QoS) is described. According to one aspect of the present invention, a method is provided for managing bandwidth allocation in a network that employs a non-deterministic access protocol. A packet forwarding device receives information indicative of a set of traffic groups. The packet forwarding device additionally receives parameters, such as bandwidth and priority parameters, corresponding to the traffic groups. After receiving a packet associated with one of the traffic groups on a first port, the packet forwarding device schedules the packet for transmission from a second port based upon parameters corresponding to the traffic group with which the packet is associated. Advantageously, in this manner, a weighted fair queuing schedule that shares bandwidth according to some set of rules may be achieved.

According to another aspect of the present invention, a method is provided for managing bandwidth allocation and traffic prioritization in a packet forwarding device. The packet forwarding device receives information indicative of a set of traffic groups. The packet forwarding device additionally receives information defining a Quality of Service (QoS) policy for the traffic groups. After a packet is received by the packet forwarding device, a traffic group with which the packet is associated is identified. Subsequently, rather than relying on an end-to-end signaling protocol for scheduling, the packet is scheduled for transmission based upon the QoS policy for the identified traffic group. Therefore, bandwidth allocation and traffic prioritization are based upon a set of administrative policies over which the network manager retains control.

According to yet another aspect of the present invention, a number of QoS queues are provided at each port of the

packet forwarding device. A current bandwidth metric is determined for each of the QoS queues for a particular port. The QoS queues are divided into two groups based upon their respective bandwidth metrics and their respective minimum bandwidth requirements. Subsequently, the groups are used as a first level arbitration mechanism to select a QoS queue that will source the next packet.

Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. **1A** is a simplified block diagram of an exemplary switch architecture in which one embodiment of the present invention may be implemented.

FIG. **1B** is a logical view of the interaction between switch processing blocks according to one embodiment of the present invention.

FIG. **2** is a flow diagram illustrating high level bandwidth management and traffic prioritization processing according to one embodiment of the present invention.

FIG. **3** is a flow diagram illustrating periodic evaluation of QoS categories according to one embodiment of the present invention.

FIG. **4** is a flow diagram illustrating next packet scheduling according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A flexible, policy-based, mechanism for managing, monitoring, and prioritizing traffic within a network and allocating bandwidth to achieve true Quality of Service (QoS) is described. "Quality of Service" in this context essentially means that there is a quantifiable measure of the service being provided. The measure of service being provided may be in terms of a packet loss rate, a maximum delay, a committed minimum bandwidth, or a limited maximum bandwidth, for example.

In the present invention, a number of QoS queues may be provided at each port of a packet forwarding device, such as a Local Area Network (LAN) switch. Based upon a set of QoS parameters, various types of traffic can be distinguished and associated with particular QoS queues. For example, packets associated with a first traffic group may be placed onto a first QoS queue and packets associated with another traffic group may be placed onto a second QoS queue. When a port is ready to transmit the next packet, a scheduling mechanism may be employed to select which QoS queue of the QoS queues associated with the port will provide the next packet for transmission.

In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

The present invention includes various steps, which will be described below. The steps of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor programmed with the instructions to perform the steps. Alternatively, the steps may be performed by a combination of hardware and software. While, embodiments of the present invention will be described with reference to a high speed Ethernet switch, the method and apparatus described herein are equally applicable to other types of network devices or packet forwarding devices.

An Exemplary Switch Architecture

An overview of the architecture of a switch **100** in which one embodiment of the present invention may be implemented is illustrated by FIG. **1A**. The central memory architecture depicted includes multiple ports **105** and **110** each coupled via a channel to a filtering/forwarding engine **115**. Also coupled to the filtering/forwarding engine **115** is a forwarding database **120**, a packet Random Access Memory (RAM) **125**, and a Central Processing Unit (CPU) **130**.

According to one embodiment, each channel is capable of supporting a data transfer rate of one gigabit per second in the transmit direction and one gigabit per second in the receive direction, thereby providing 2 Gb/s full-duplex capability per channel. Additionally, the channels may be configured to support one Gigabit Ethernet network connection or eight Fast Ethernet network connections.

The filtering/forwarding engine **115** includes an address filter (not shown), a switch matrix (not shown), and a buffer manager (not shown). The address filter may provide bridging, routing, Virtual Local Area Network (VLAN) tagging functions, and traffic classification. The switch matrix connects each channel to a central memory such as packet RAM **125**. The buffer manager controls data buffers and packet queue structures and controls and coordinates accesses to and from the packet RAM **125**.

The forwarding database **120** may store information useful for making forwarding decisions, such as layer **2** (e.g., Media Access Control (MAC) layer), layer **3** (e.g., Network layer), and/or layer **4** (e.g., Transport layer) forwarding information, among other things. The switch **100** forwards a packet received at an input port to an output port by performing a search on the forwarding database using address information contained within the header of the received packet. If a matching entry is found, a forwarding decision is constructed that indicates to which output port the received packet should be forwarded, if any. Otherwise, the packet is forwarded to the CPU **130** for assistance in constructing a forwarding decision.

The packet RAM **125** provides buffering for packets and acts as an elasticity buffer for adapting between incoming and outgoing bandwidth differences. Packet buffering is discussed further below.

Logical View of Exemplary Switch Processing

FIG. **1B** is a logical view of the interaction between exemplary switch processing blocks that may be distributed throughout the switch **100**. For example, some of the processing may be performed by functional units within the ports of the switch and other processing may be performed by the CPU **130** or by the address filter/switch matrix/buffer manager **115**. In any event, the processing can be conceptually divided into a first group of functions **160** dedicated to input processing and a second group of functions **185** dedicated to output processing. According to the present embodiment, the first group **160** includes a comparison engine **155**, an enqueue block **161**, a packet classification block **150**, and a buffer manager **165**. The second group **185** includes a dequeue block **162**, a Quality of Service (QoS) category evaluation block **175**, and a scheduler **170**.

Additionally, a user interface (UI) **145** may be provided for receiving various parameters from the network manager. The UI may be text based or graphical. In one embodiment, the UI **145** may include an in-band HyperText Markup Language (HTML) browser-based management tool which may be accessed by any standard web browser. In any event, the goal of the UI **145** is to separate high-level policy components, such as traffic grouping and QoS profiles from the details of the internal switch hardware. Thus, user configuration time is minimized and a consistent interface is provided to the user.

The UI **145** receives information indicative of one or more traffic groups. This information may be provided by the network manager. There are several ways to define a traffic group. Table 1 below illustrates a variety of traffic classification schemes that may be supported by the UI **145**.

TABLE 1

Traffic Classification		
Policy Based Upon	Traffic Group Definition	OSI Layer
Applications	TCP Session UDP Session RSVP Flow	Transport Layer
Network Layer Topology or Groups of Users	Network Layer Protocol Subnet or IP Address VLAN Identifier	Network Layer
End-Station Applications	MAC Address 802.1p or 802.1Q	Link Layer
Physical Topology	Physical Port	Physical Layer

The information used to identify a traffic group typically depends upon what terms the QoS policy is defined. If the QoS policy is based on applications, traffic groups may be differentiated at the Transport layer by Transmission Control Protocol (TCP) session or User Datagram Protocol (UDP) session. For example, the network manager may provide information indicative of TCP source and destination ports and IP source and destination addresses to identify traffic groups. However, if the QoS policy is based upon the Network layer topology or groups of users, traffic group definition may be more convenient by supplying information regarding the Network layer protocol, such as Internet Protocol (IP) or Internetwork Packet Exchange (IPX), the subnet or IP addresses, or VLAN identifiers. If the QoS policy is defined by end-station applications, then Media Access Control (MAC) addresses, IEEE 802.1p priority indications, or IEEE 802.1Q frames may be employed to identify traffic groups. Finally, if the QoS policy is physical topology based, physical port identifiers may be used to differentiate traffic groups.

It should be noted that Table 1 merely presents an exemplary set of traffic group identification mechanisms. From the examples presented herein, additional, alternative, and equivalent traffic grouping schemes and policy considerations will be apparent to those of ordinary skill in the art. For example, other state information may be useful for purposes of packet classification, such as the history of previous packets, the previous traffic load, the time of day, etc.

It is appreciated that traffic classifications based upon the traffic group definitions listed above may result in overlap. Should the network manager define overlapping traffic groups, the UI **145** may issue an error message and reject the most recent traffic group definition, the UI **145** may issue a warning message to the network manager and allow the more specific traffic group definition to override a conflicting general traffic group definition, or the UI **145** may be configured to respond in another manner.

A number of QoS queues **180** may be provided at each of the ports of a packet forwarding device. In one embodiment, a mapping of traffic groups to QoS queues **180** may be maintained. As traffic groups are provided by the network manager, the UI **145** updates the local mapping of traffic groups to QoS queues **180**. This mapping process may be a one-to-one mapping of the traffic groups defined by the network manager to the QoS queues **180** or the mapping process may be more involved. For example, there may be more traffic groups than QoS queues **180**, in which case, more than one traffic group will be mapped to a single QoS queue. Some consolidation rules for combining multiple traffic groups into a single QoS queue will be discussed below.

At any rate, by providing a layer of abstraction in this manner, the network manager need not be burdened with the underlying implementation details, such as the number of QoS queues per port and other queuing parameters. Another advantage achieved by this layer of abstraction between the traffic group definitions and the physical QoS queues is the fact that the UI **145** is now decoupled from the underlying implementation. Therefore, the UI **145** need not be updated if the hardware QoS implementation changes. For example, software providing for traffic group definition need not be changed simply because the number of QoS queues per port provided by the hardware changes.

The input data stream is received by the comparison engine **155** from input switch ports (not shown). Under the direction of the packet classification process **150**, the comparison engine **155** determines with which of the previously defined traffic groups a packet in the data stream is associated. The packet classification block **150** may employ the traffic group indications provided by the network manager to provide the comparison engine **155** with information regarding locations and fields to be compared or ignored within the header of a received packet, for example. It should be appreciated if the comparison required for traffic classification is straightforward, such as in a conventional packet forwarding device, then the comparison engine **155** and the packet classification block **150** may be combined.

The packet classification block **150** in conjunction with the UI **145** provide a network manager with a flexible mechanism to control traffic prioritization and bandwidth allocation through the switch **100**. Importantly, no end-to-end signaling protocol needs to be implemented by the network devices. For example, the end-station that is to receive the data stream need not reserve bandwidth on each of the intermediate devices between it and the source of the data stream. Rather, a packet forwarding device employing the present invention can provide some benefit to the network without requiring routers and/or end-stations to do anything in particular to identify traffic. Thus, traffic priority may be enforced by the switch **100** and QoS may be delivered to applications without altering routers or end-stations.

According to one embodiment, the buffer manager **165** participates in policy based QoS by controlling the allocation of buffers within the packet RAM **125**. Buffers may be dynamically allocated to QoS queues **180** as needed, within constraints established by QoS profile attributes, which are discussed below. The buffer manager **165** may maintain several programmable variables for each QoS queue. For example, a Minimum Buffer Allocation and a Maximum Queue Depth may be provided for each QoS queue. The Minimum Buffer Allocation essentially reserves some minimum number of buffers in the packet RAM **125** for the QoS queue with which it is associated. The Maximum Queue

Depth establishes the maximum number of buffers that can be placed on a given QoS queue. The buffer manager **165** also maintains a Current Queue Depth for each QoS queue to assure the maximum depth is not exceeded. For example, before allowing a buffer to be added to a given QoS queue, the buffer manager **165** may compare the Maximum Queue Depth to the Current Queue Depth to ensure the Maximum Queue Depth is not exceeded.

Variables are also maintained for tracking free buffers in the packet RAM **125**. At initialization, a Buffers Free Count contains the total number of buffers available in the packet RAM **125** and a Buffers Reserved Count contains the sum of the minimum buffer allocations for the QoS queues **180**. As packets are received they are stored in free buffers, and the Buffers Free Count is decremented by the number of buffers used for such storage. After the appropriate QoS queue has been identified the buffer manager **165** instructs the enqueue block **161** to add the packet to the QoS queue. The enqueue block **161** links the packet to the identified queue provided that the Current Queue Depth is less than the Maximum Queue Depth and either (1) the Current Queue Depth is less than the Minimum Buffer Allocation or (2) the Buffers Reserved Count is less than the Buffers Free Count. Therefore, if a QoS queue exceeds its reserve of buffers (e.g., Minimum Buffer Allocation), to the extent that additional buffers remain free, the QoS queue may continue to grow. Otherwise, the enqueue block **161** will discard the packet, the buffers are returned to the free pool, and the Buffers Free Count is increased by the number of buffers that would have been consumed by the packet. When a packet is successfully linked to a QoS queue, the Current Queue Depth for that QoS queue is increased by the number of buffers used by the packet. If, prior to the addition of the packet to the queue, the Current Queue Depth was less than the Minimum Buffer Allocation then the Buffers Reserved Count is decreased by the lesser of (1) the number of buffers in the packet or (2) the difference between the Current Queue Depth and the Minimum Buffer Allocation.

The QoS category evaluation process **175** separates the QoS queues into a plurality of categories based upon a set of bandwidth parameters. The scheduler **170** uses the grouping provided by the QoS category evaluation process **175** to select an appropriate QoS queue for sourcing the next packet for a particular port. The evaluation of QoS queue categories may be performed periodically or upon command by the scheduler **170**, for example. Periodic evaluation of QoS categories and scheduling is discussed in further detail below.

Responsive to the scheduler **170** the dequeue block **162** retrieves a packet from a specified QoS queue. After the packet has been transmitted, the buffer variables are updated. The Buffers Free Count is increased and the Current Queue Depth is decreased by the number of buffers utilized to store the packet. If the resulting Current Queue Depth is less than the Minimum Buffer Allocation, then the Buffers Reserved Count is increased by the lesser of the number of buffers utilized to store the packet or the difference between the Current Queue Depth and the Minimum Buffer Allocation.

QoS Profile Attributes

Setting QoS policy is a combination of identifying traffic groups and defining QoS profiles for those traffic groups. According to one embodiment, each individual traffic group may be associated with a QoS profile. However, in alternative embodiments, multiple traffic groups may share a common QoS profile. Having described traffic group classification and identification above, QoS profile attributes (also referred to as parameters) will now be discussed.

Several queuing mechanisms may be implemented using one or more of the following parameters associated with a traffic group: (1) minimum bandwidth, (2) maximum bandwidth, (3) peak bandwidth, (4) maximum delay, and (5) relative priority. In general, the minimum, maximum, and peak bandwidth parameter may be expressed in Mbps, a percentage of total bandwidth, or any other convenient representation.

Minimum bandwidth indicates the minimum amount of bandwidth a particular traffic group needs to be provided over a defined time period. If the sum of the minimum bandwidths for all traffic groups defined is less than 100% of the available bandwidth, then the scheduling processing, discussed below, can assure that each traffic group will receive at least the minimum bandwidth requested.

Maximum bandwidth is the maximum sustained bandwidth the traffic group can realize over a defined time period. In contrast, peak bandwidth represents the bandwidth a traffic group may utilize during a particular time interval in excess of the maximum bandwidth. The peak bandwidth parameter may be used to limit traffic bursts for the traffic group with which it is associated. The peak bandwidth also determines how quickly the traffic group's current bandwidth will converge to the maximum bandwidth. By providing a peak bandwidth value that is much higher than the maximum bandwidth, if sufficient bandwidth is available, the maximum bandwidth will be achieved relatively quickly. In contrast, a peak bandwidth that is only slightly higher that the maximum bandwidth will cause the convergence to the maximum bandwidth to be more gradual.

Maximum delay specifies a time period beyond which further delay cannot be tolerated for the particular traffic group. Packets comprising the traffic group that are forwarded by the switch **100** are guaranteed not to be delayed by more than the maximum delay specified.

Relative priority defines the relative importance of a particular traffic group with respect to other traffic groups. As will be discussed further below, within the same QoS category, traffic groups with a higher priority are preferred over those with lower priorities.

This small set of parameters in combination with the variety of traffic classification schemes gives a network manager enormous control and flexibility in prioritizing and managing traffic flowing through packet forwarding devices in a network. For example, the QoS profile of a video traffic group, identified by UDP session, might be defined to have a high priority and a minimum bandwidth of 5 Mbps, while the QoS profile of an engineering traffic group, identified by VLAN, may be set to a second priority, a minimum bandwidth of 30 Mbps, a maximum bandwidth of 50 Mbps, and a peak bandwidth of 60 Mbps. Concurrently, the QoS profile of a World Wide Web (WWW) traffic group, identified by protocol (e.g., IP), may be set to have a low priority, a minimum bandwidth of 0 Mbps, a maximum bandwidth of 100%, and a peak bandwidth of 100%.

Consolidation Rules

It was mentioned earlier that multiple traffic groups may be mapped to a single QoS queue. This may be accomplished by maintaining an independent set of variables (e.g., minimum bandwidth, maximum bandwidth, peak bandwidth, maximum delay, and relative priority) for each QoS queue in addition to those already associated with each traffic group and following the general consolidation rules outlined below.

Briefly, when the mapping from traffic groups to QoS queues is one-to-one, the determination of a particular QoS queues' attributes is straightforward. The QoS queue's

attributes simply equal the traffic group's attributes. However, when combining multiple traffic groups that do not share a common QoS profile onto a single QoS queue, the following general consolidation rules are suggested: (1) add minimum attributes of the traffic groups being combined to arrive at an appropriate minimum attribute for the target QoS queue (e.g., the QoS queue in which the traffic will be merged), (2) use the largest of maximum attributes to arrive at an appropriate value for a maximum attribute for the target QoS queue, and (3) avoid merging traffic groups that have different relative priorities. This last rule suggests the number of priority levels provided should be less than or equal to the number of QoS queues supported by the implementation to assure traffic groups with different priorities are not combined in the same QoS queue.

Importantly, when a network manager has determined that multiple traffic groups will share a common QoS profile, the consolidation rules need not apply, as the network manager has already, in effect, manually consolidated the parameters.

Bandwidth Management and Traffic Prioritization

Having described an exemplary environment in which one embodiment of the present invention may be implemented, bandwidth management and traffic prioritization will now be described with reference to FIG. **2**. FIG. **2** is a flow diagram illustrating the high level bandwidth management and traffic prioritization processing according to one embodiment of the present invention. In this embodiment, at step **210**, a manager-defined QoS policy may be received via the UI **145**, for example. The QoS policy is a combination of traffic groups and QoS profile attributes corresponding to those traffic groups.

At step **220**, a packet is received by the switch **100**. Before the packet can be placed onto a QoS queue for transmission, the traffic group to which the packet belongs is identified at step **230**. Typically, information in the packet header, for example, can be compared to the traffic group criteria established by the network manager to identify the traffic group to which the packet belongs. This comparison or matching process may be achieved by programming filters in the switch **100** that allow classification of traffic. According to one embodiment, the packet may be identified using the traffic group definitions listed in Table 1.

At step **250**, enqueue processing is performed. The packet is added to the rear of the appropriate QoS queue for the identified traffic group. Importantly, if a maximum delay has been assigned to the traffic group with which the packet is associated, then the packet should either be dropped or transmitted within the period specified. According to one embodiment, this may be accomplished by limiting the depth (also referred to as length) of the corresponding QoS queue. Given the minimum bandwidth of the QoS queue and the maximum delay the traffic group can withstand, a maximum depth for the QoS queue can be calculated. If the QoS queue length remains less than or equal to the maximum length, then the packet is added to the QoS queue. However, if the QoS queue length would exceed the maximum length by the addition, then the packet is dropped.

At step **260**, scheduling is performed. The scheduling/dequeuing processing involves determining the appropriate QoS queue group, selecting the appropriate QoS queue within that QoS queue group, and removing the packet at the front of the selected QoS queue. This selected packet will be the next packet the port transmits. Scheduling will be discussed further below.

Evaluation of QoS Categories

According to one embodiment of the present invention, it is advantageous to divide the QoS queues into at least two categories. The categories may be defined based upon the maximum bandwidth, the minimum bandwidth, the peak bandwidth, and the "current bandwidth." The current bandwidth should not be mistaken for a bandwidth at an instant in time, rather the current bandwidth is a moving average that is updated periodically upon the expiration of a predetermined time period. Empirical data suggests this predetermined time period should be on the order of ten packet times, wherein a packet time is the time required to transmit a packet. However, depending upon the environment and the nature of the traffic, a value in the range of one to one hundred packet times may be more suitable.

The members of the first category ("Category A") are those QoS queues which have a current bandwidth that is below their peak bandwidth and below their minimum bandwidth. Members of the second category ("Category B") include those QoS queues that have a current bandwidth that is greater than or equal to their minimum bandwidth, but less than both their maximum bandwidth and their peak bandwidth. The remaining QoS queues (e.g., those having a current bandwidth that is greater than or equal to either the peak bandwidth or the maximum bandwidth) are ineligible for transmission. These QoS queues that are ineligible for transmission can be considered a third category ("Category C"). With this overview of QoS categories, an exemplary process for periodic evaluation of QoS categories will now be described.

FIG. **3** is a flow diagram illustrating periodic evaluation of QoS categories according to one embodiment of the present invention. In this embodiment, at step **310**, processing loops until the predetermined evaluation time period has expired. For example, a test may be performed to determine if the current time is greater than or equal to the last evaluation time plus the predetermined evaluation time interval. Alternatively, the evaluation process may be triggered by an interrupt. In any event, when it is time to evaluate the QoS queue categorization, processing continues with step **320**.

It will be appreciated that the time interval chosen for the predetermined evaluation time period should not be too long or too short. If the time interval is too long, one QoS queue might be allowed to monopolize the link until its maximum bandwidth is achieved while other QoS queues remain idle. If the time interval is too short, transmitting a single packet or remaining idle for a single packet time may cause the QoS queue to become a member of a different QoS category (e.g., the single transmission may cause the current bandwidth to exceed the maximum bandwidth or the single idle time may cause the current bandwidth to fall below the minimum bandwidth) because the moving average moves very quickly over short time intervals.

At step **330**, the current bandwidth for a particular QoS queue is set to the current bandwidth for that QoS queue as calculated in the previous time interval multiplied by a first weighting factor plus the actual bandwidth that particular QoS queue received in the previous time interval multiplied by a second weighting factor, wherein the weighting factors may be selected to achieve the desired level of responsiveness in the current bandwidth metric. For example, it may be desirable to have the current bandwidth converge to within a certain percentage of a sustained bandwidth if that bandwidth has been sustained for a certain amount of time. Exemplary weighting factors are in the form (w−1)/w and 1/w, respectively. Using weighting factors of 15/16 for the first weighting factor and a value of 1/16 for the second weighting factor, for example, the current bandwidth will reflect 50% of a step within 13 time intervals, 80% of a step within 27 time intervals, and will be within 2% of the

sustained bandwidth in approximately 63 time intervals (assuming a maximum and peak bandwidth of 100%). Alternative ratios and current bandwidth metrics will be apparent to those of ordinary skill in the art.

After the current bandwidth has been evaluated for a QoS queue, at step **340**, the QoS queue bandwidth parameters can be compared to the current bandwidth to determine to which QoS category the QoS queue belongs. As described above, if (CURR_BW<PEAK_BW) and (CURR_BW<MIN_BW), then the QoS queue is associated with Category A at step **350**. If (CURR_BW≧MIN_BW) and ((CURR_BW<MAX_BW) and (CURR_BW<PEAK_BW)), then the QoS queue is associated with Category B at step **360**. If (CURR_BW≧PEAK_BW) or (CURR_BW≧MAX_BW), then the QoS queue is associated with Category C at step **370**.

At step **380**, if all of the QoS queues have been evaluated, then processing branches to step **310**; otherwise, processing continues with step **330**.

Scheduling Processing

Briefly, at each port, three levels of arbitration may be employed to select the appropriate QoS queue from which to transmit the next packet. The first level of arbitration selects among the QoS categories. Category A is given priority if any member QoS queues have one or more pending packets. Otherwise, a QoS queue with one or more pending packets of Category B is selected. According to one embodiment, the relative priority assigned to each QoS queue may be used as a second level of arbitration. In this manner, when multiple QoS queues satisfy the first level arbitration, a higher priority QoS queue is favored over a lower priority QoS queue. Finally, when there is a tie at the second level of arbitration (e.g., two or more QoS queues in the same QoS category have the same relative priority), a round robin or least recently used (LRU) scheme may be employed to select from among the two or more QoS queues until the QoS categories are evaluated.

Assuming a periodic evaluation of QoS categories is being performed, the scheduling processing need not include such evaluation and the scheduling processing may be performed as illustrated by FIG. **4**, according to one embodiment of the present invention. In the embodiment depicted, at step **410**, processing loops until the port associated with the group of QoS queues being evaluated indicates it is ready to receive the next packet for transmission. For example, the port may be polled to determine its transmission status. Alternatively, the scheduling process may be triggered by an interrupt. In any event, when the port is ready for the next packet, processing continues with step **420**.

At step **420**, a QoS category is selected from which a QoS queue will provide the next packet for transmission. As described above, priority is given to the category containing QoS queues with pending data that are below the peak bandwidth and minimum bandwidth (e.g., Category A). However, if no QoS queues meet this criteria, Category B is selected.

At step **430**, if multiple QoS queues are members of the selected QoS category, processing continues with step **440**; otherwise, processing branches to step **470**.

At step **440**, the relative priorities of the QoS queues are used to select among the QoS queues of the selected category that have pending data.

At step **450**, if two or more QoS queues have the same priority, then processing continues with step **460**. Otherwise, if a QoS queue is found to have the highest relative priority, then processing branches to step **470**.

At step **460**, the tie is resolved by performing round robin or LRU scheduling. That is, until the QoS categories are evaluated, the QoS queues having the same priority will be rotated through in a predetermined order or scheduled such that the QoS queue that has not provided a packet for transmission recently will be given such an opportunity. After selecting a QoS queue in this manner, processing continues with step **470**.

At step **470**, a packet is dequeued from the selected QoS queue and the packet is transmitted by the port at step **480**. This scheduling process may be repeated by looping back to step **410**, as illustrated.

Queuing Schemes

A variety of different queuing mechanisms may be implemented using various combinations of the QoS profile attributes discussed above. Table 2 below illustrates how to achieve exemplary queuing mechanisms and corresponding configurations of the QoS profile attributes.

TABLE 2

Queuing Mechanism Configurations

Queuing Mechanism	QoS Profile Attribute Value
Strict Priority Queuing	Minimum Bandwidth = 0% Maximum Bandwidth = 100% Peak Bandwidth = 100% Maximum Delay = N/A Relative Priority = $PRIORITY_i$
Round Robin/ Least Recently Used Queuing	Minimum Bandwidth = 0% Maximum Bandwidth = 100% Peak Bandwidth = 100% Maximum Delay = N/A Relative Priority = <same for all queues>
Weighted Fair Queuing	Minimum Bandwidth = >0% Maximum Bandwidth = MAX_BW_i Peak Bandwidth = $PEAK\_BW_i$ Maximum Delay = N/A Relative Priority = <same for all queues>

$PRIORITY_i$ represents a programmable priority value for a particular QoS queue, i. Similarly, MAX_BW_i and $PEAK\_BW_i$ represent programmable maximum bandwidths and peak bandwidths, respectively, for a particular QoS queue, i.

For a strict priority scheme, each QoS queue's minimum bandwidth is set to zero percent, each QoS queue's maximum bandwidth is set to one hundred percent, and each QoS queue's peak is set to one hundred percent. In this manner, the current bandwidth will never be less than the minimum bandwidth, and the current bandwidth will never exceed either the peak bandwidth or the maximum bandwidth. In this configuration, all QoS queues will be associated with Category B since no QoS queues will satisfy the criteria of either Category A or Category B. Ultimately, by configuring the QoS profile attributes in this manner, the second level of arbitration (e.g., the relative priority of the QoS queues) determines which QoS queue is to source the next packet.

For a pure round robin or least recently used (LRU) scheme, the QoS profile attributes are as above, but additionally all QoS queue priorities are set to the same value. In this manner, the third level of arbitration determines which QoS queue is to source the next packet.

Finally, weighted fair queuing can be achieved by assigning, at least, a value greater than zero percent to the desired minimum bandwidth. By assigning a value greater than zero to the minimum bandwidth parameter, the particular QoS queue is assured to get at least that amount of bandwidth on average because the QoS queue will be associated with Category A until at least its minimum

bandwidth is satisfied. Additionally, different combinations of values may be assigned to the peak and maximum bandwidths to prevent a particular QoS queue from monopolizing the link.

Alternative Embodiments

While evaluation of QoS categories has been described above as occurring periodically, this evaluation may also be triggered by the occurrence of a predetermined event. Alternatively, evaluation of QoS categories may take place as part of the scheduling processing rather than as part of a separate periodic background process.

While a relationship between the number of priority levels and the number of QoS queues has been suggested above, it is appreciated that the number of QoS queues may be determined independently of the number of priority levels. Further, it is appreciated that the number of QoS queues provided at each port may be fixed for every port or alternatively a variable number of QoS queues may be provided for each port.

Finally, in alternative embodiments, weighting factors and ratios other than those suggested herein may be used to adjust the current bandwidth calculation for a particular implementation.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method comprising:

receiving at a packet forwarding device information indicative of one or more traffic groups;

receiving at the packet forwarding device one or more bandwidth parameters for at least one of the one or more traffic groups;

receiving at a first port of a plurality of ports a packet associated with the at least one traffic group;

enqueuing the packet onto a queue associated with the at least one traffic group;

scheduling the packet for transmission from a second port of the plurality of ports based upon the one or more bandwidth parameters for the at least one traffic group with which the packet is associated; by

periodically evaluating a current bandwidth metric for the queue; by

determining an actual bandwidth for a prior time period;

determining a bandwidth metric for the prior time period; and

combining a portion of the actual bandwidth for the prior time period with a portion of the bandwidth metric for the prior time period to arrive at the current bandwidth metric; and

dequeuing the packet from the queue if the current bandwidth metric meets a predetermined relationship with the one or more bandwidth parameters.

2. The method of claim 1, wherein the network employs a non-deterministic access protocol.

3. The method of claim 2, wherein the non-deterministic access protocol is Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

4. The method of claim 1, wherein the one or more bandwidth parameters include an indication regarding a minimum amount of bandwidth the at least one traffic group needs to be provided over a defined time period.

5. The method of claim 1, wherein the one or more bandwidth parameters include an indication regarding a maximum sustained bandwidth the at least one traffic group can realize over a defined time period.

6. The method of claim 5, wherein the one or more bandwidth parameters include an indication regarding a peak bandwidth representing a bandwidth the at least one traffic group may utilize during a particular time interval in excess of the maximum bandwidth.

7. The method of claim 1, further comprising: classifying the packet as being associated with the at least one traffic group; and determining a quality of service queue with which the at least one traffic group is associated.

8. The method of claim 1, wherein QoS profile attributes associated with each of the one or more traffic groups include a maximum delay, specifying a time period beyond which further delay cannot be tolerated for the particular traffic group.

9. The method of claim 1, wherein the other QoS profile attributes associated with each of the one or more traffic groups include a Relative Priority, defining the relative importance of a particular traffic group with respect to other traffic groups.

10. A method of bandwidth management and traffic prioritization for use in a network of devices, the method comprising the steps of:

defining at a packet forwarding device information indicative of one or more traffic groups;

defining at the packet forwarding device information indicative of a quality of service (QoS) policy for at least one of the one or more traffic groups;

receiving a packet at a first port of a plurality of ports;

identifying a first traffic group of the one or more traffic groups with which the packet is associated;

providing a plurality of QoS queues;

mapping the first traffic group to a first QoS queue of the plurality of QoS queues; and

scheduling the packet for transmission from a second port of the plurality of ports based upon the QoS policy for the first traffic group, and wherein the scheduling is independent of end-to-end signaling; said scheduling including:

determining a current bandwidth metric for each of the plurality of QoS queues;

dividing the plurality of QoS queues into at least a first group and a second group based upon the current bandwidth metrics and a minimum bandwidth requirement associated with each of the plurality of QoS queues; and

if the first group includes at least one QoS queue, then transmitting a packet from the at least one QoS queue; otherwise transmitting a packet from a QoS queue associated with the second group.

11. The method of claim 10, wherein the network of devices employs a non-deterministic access protocol.

12. The method of claim 11, wherein the non-deterministic access protocol is Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

13. A method of bandwidth management and traffic prioritization for use in a network of devices, the method comprising:

receiving at a packet forwarding device information indicative of one or more traffic groups, the information indicative of the one or more traffic groups including Internet Protocol (IP) subnet membership;

receiving at the packet forwarding device information defining a quality of service (QoS) policy for at least

one of the one or more traffic groups, the QoS policy including at least a minimum bandwidth;

providing a plurality of queues at each of a plurality of output ports;

associating the one or more traffic groups with the plurality of queues based upon the minimum bandwidth; and

scheduling a packet for transmission from one of the plurality of queues onto the network.

14. A method of bandwidth management and traffic prioritization for use in a network of devices, the method comprising:

providing a plurality of quality of service (QoS) queues at each of a plurality of output ports, each of the plurality of QoS queues associated with a minimum queue bandwidth requirement;

adding a packet to one of the plurality of QoS queues based upon a traffic group with which the packet is associated; and

scheduling a next packet for transmission onto the network from one of the plurality of QoS queues at a particular output port of the plurality of output ports by:

determining a current bandwidth metric for each of the plurality of QoS queues,

dividing the plurality of QoS queues into at least a first group and a second group based upon the current bandwidth metrics and the minimum queue bandwidth requirements, and

if at least one QoS queue of the plurality of QoS queues, so divided, is associated with the first group, then transmitting a packet from the at least one QoS queue; otherwise transmitting a packet from a QoS queue of the plurality of QoS queues associated with the second group.

15. The method of claim **14**, wherein the current bandwidth for a particular QoS queue is calculated as follows:

$$CURR\_BW_t = W1 \times CURR\_BW_t + W2 \times ACT\_BW_t;$$

where:

$CURR\_BW_t$ represents the current bandwidth for a particular QoS queue,

W1 represents a first weighting factor,

W2 represents a second weighting factor, and

ACT_BW_t represents the actual bandwidth received by the particular QoS queue in a previous time interval.

16. The method of claim **15**, wherein W1=(W−1)/W, W2=1/W, and the previous time interval is the most recent time interval.

17. The method of claim **14**, further comprising selecting an appropriate QoS queue, from which to transmit the next packet, from the first or the second group based upon relative queue priorities associated with the QoS queues.

18. The method of claim **14**, further comprising selecting an appropriate QoS queue, from which to transmit the next packet, from the first or the second group based upon a round robin selection scheme.

19. The method of claim **14**, further comprising selecting the appropriate QoS queue, from which to transmit the next packet, from the first or the second group based upon a least recently used (LRU) selection scheme.

20. The method of claim **14**, wherein the first group comprises QoS queues associated with a minimum queue bandwidth requirement that is less than the QoS queues' current bandwidth metric, and wherein the second group comprises QoS queues associated with a minimum queue bandwidth requirement that is greater than or equal to the QoS queues' current bandwidth metric.

21. A packet forwarding device for use in a network employing a non-deterministic assess protocol, the packet forwarding device comprising:

a filtering and forwarding engine configured to forward received packets based upon a traffic group with which the packet is associated; and

a plurality of ports coupled to the filtering and forwarding engine, each port of the plurality of ports configured to receive packets from the filtering and forwarding engine, each port of the plurality of ports having a plurality of Quality of Service (QoS) queues associated with a minimum queue bandwidth requirement, each port of the plurality of ports further configured to schedule a packet for transmission onto the network by determining a current bandwidth metric for each of the plurality of QoS queues,

dividing the plurality of QoS queues into at least a first group and a second group based upon the current bandwidth metrics and the minimum queue bandwidth requirements, and

if at least one QoS queue of the plurality of QoS queues, so divided, is associated with the first group, then transmitting a packet from the at least one QoS queue; otherwise transmitting a packet from a QoS queue of the plurality of QoS queues associated with the second group.

22. The packet forwarding device of claim **21**, wherein the plurality of ports are further configured to select among QoS queues in the same group based upon relative queue priorities associated with the QoS queues.

23. The packet forwarding device of claim **21**, wherein the plurality of ports are further configured to select among QoS queues in the same group based upon a round robin selection scheme.

24. The packet forwarding device of claim **21**, wherein the plurality of ports are further configured to select among QoS queues in the same group based upon a least recently used (LRU) selection scheme.

25. The packet forwarding device of claim **21**, wherein the first group comprises QoS queues associated with a minimum queue bandwidth requirement that is less than the QoS queues' current bandwidth metric, and wherein the second group comprises QoS queues associated with a minimum queue bandwidth requirement that is greater than or equal to the QoS queues' current bandwidth metric.

26. A method of bandwidth management and traffic prioritization for use in a network of devices, the method comprising:

receiving at a packet forwarding device information indicative of one or more traffic groups, the information indicative of the one or more traffic groups including a virtual local area network (VLAN) identifier;

receiving at the packet forwarding device information defining a quality of service (QoS) policy for at least one of the one or more traffic groups, the QoS policy including at least a minimum bandwidth;

providing a plurality of queues at each of a plurality of output ports;

associating the one or more traffic groups with the plurality of queues based upon the minimum bandwidth; and

scheduling a packet for transmission from one of the plurality of queues onto the network.

27. A machine-readable medium having stored thereon data representing sequences of instructions, said sequences

of instructions which, when executed by a processor, cause said processor to:

receive at a packet forwarding device information indicative of one or more traffic groups;

receive at the packet forwarding device one or more bandwidth parameters for at least one of the one or more traffic groups;

receive at a first port of a plurality of ports a packet associated with the at least one traffic group;

enqueue the packet onto a queue associated with the at least one traffic group;

schedule the packet for transmission from a second port of the plurality of ports based upon the one or more bandwidth parameters for the at least one traffic group with which the packet is associated; by

periodically evaluating a current bandwidth metric for the queue; by

determining an actual bandwidth for a prior time period;

determining a bandwidth metric for the prior time period; and

combining a portion of the actual bandwidth for the prior time period with a portion of the bandwidth metric for the prior time period to arrive at the current bandwidth metric; and

dequeuing the packet from the queue if the current bandwidth metric meets a predetermined relationship with the one or more bandwidth parameters.

28. A machine-readable medium having stored thereon data representing sequences of instructions, said sequences of instructions which, when executed by a processor, cause said processor to:

define at a packet forwarding device information indicative of one or more traffic groups;

define at the packet forwarding device information indicative of a quality of service (QoS) policy for at least one of the one or more traffic groups;

receive a packet at a first port of a plurality of ports;

identify a first traffic group of the one or more traffic groups with which the packet is associated;

provide a plurality of QoS queues;

map the first traffic group to a first QoS queue of the plurality of QoS queues; and

schedule the packet for transmission from a second port of the plurality of ports based upon the QoS policy for the first traffic group, and wherein the scheduling is independent of end-to-end signaling; said scheduling including:

determining a current bandwidth metric for each of the plurality of QoS queues;

dividing the plurality of QoS queues into at least a first group and a second group based upon the current bandwidth metrics and a minimum bandwidth requirement associated with each of the plurality of QoS queues; and

if the first group includes at least one QoS queue, then transmitting a packet from the at least one QoS queue; otherwise transmitting a packet from a QoS queue associated with the second group.

29. A machine-readable medium having stored thereon data representing sequences of instructions, said sequences of instructions which, when executed by a processor, cause said processor to:

provide a plurality of quality of service (QoS) queues at each of a plurality of output ports, each of the plurality of QoS queues associated with a minimum queue bandwidth requirement;

add a packet to one of the plurality of QoS queues based upon a traffic group with which the packet is associated; and

schedule a next packet for transmission onto the network from one of the plurality of QoS queues at a particular output port of the plurality of output ports by:

determining a current bandwidth metric for each of the plurality of QoS queues,

dividing the plurality of QoS queues into at least a first group and a second group based upon the current bandwidth metrics and the minimum queue bandwidth requirements, and

if at least one QoS queue of the plurality of QoS queues, so divided, is associated with the first group, then transmitting a packet from the at least one QoS queue; otherwise transmitting a packet from a QoS queue of the plurality of QoS queues associated with the second group.

* * * * *

US006459682B1

(12) **United States Patent**
Ellesson et al.

(10) **Patent No.: US 6,459,682 B1**
(45) **Date of Patent: Oct. 1, 2002**

(54) **ARCHITECTURE FOR SUPPORTING SERVICE LEVEL AGREEMENTS IN AN IP NETWORK**

(75) Inventors: **Edward James Ellesson**, Apex, NC (US); **Roch Andre Guerin**, Yorktown Heights, NY (US); **Sanjay Damodar Kamat**, Ossining, NY (US); **Arvind Krishna**, Somers, NY (US); **Rajendran Rajan**, Bronx, NY (US); **Dinesh Chandra Verma**, Millwood, NY (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/056,526**

(22) Filed: **Apr. 7, 1998**

(51) **Int. Cl.**[7] **H04L 12/26**

(52) **U.S. Cl.** **370/235**; 370/253; 370/401; 370/411; 370/468

(58) **Field of Search** ..... 370/230, 232, 370/235, 253 R, 395, 352, 355, 356, 401, 396, 389, 231, 236, 409, 400, 412, 414, 429, 411, 238, 468; 709/232, 233

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,539,745	A *	7/1996	Chang et al.	370/85.13
5,631,905	A *	5/1997	Yano	370/462
5,812,527	A *	9/1998	Kline et al.	370/232
5,850,395	A *	12/1998	Hauser et al.	370/398
5,898,669	A *	4/1999	Shimony et al.	370/232
6,021,263	A *	2/2000	Kujoory et al.	395/200.62
6,041,053	A *	3/2000	Douceur et al.	370/389
6,041,166	A *	3/2000	Hart et al.	395/200.68
6,044,062	A *	3/2000	Brownigg et al.	370/238
6,052,734	A *	4/2000	Ito et al.	370/235
6,094,435	A *	7/2000	Hoffman et al.	370/414
6,097,722	A *	8/2000	Graham et al.	370/396
6,104,700	A *	8/2000	Haddock et al.	370/235
6,115,373	A *	9/2000	Lea	370/355

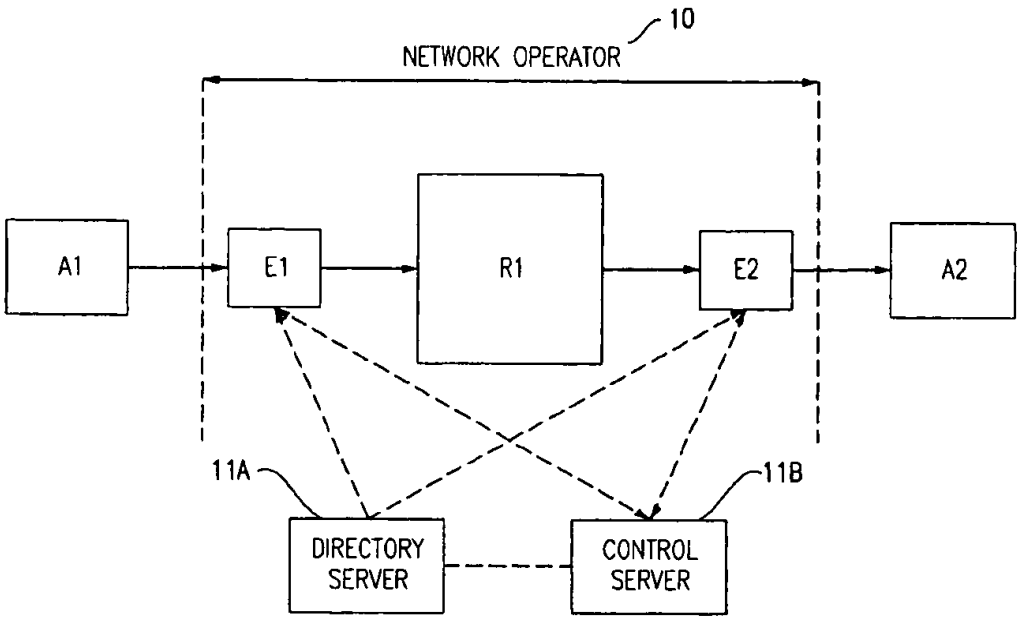
* cited by examiner

Primary Examiner—Ricky Ngo
Assistant Examiner—Phuc Tran
(74) *Attorney, Agent, or Firm*—Whitham, Curtis & Christofferson, P.C.; Manny W. Schecter

(57) **ABSTRACT**

A method of controlling packet traffic in an IP network of originating, receiving and intermediate nodes to meet performance objectives established by service level agreements. Traffic statistics and performance data such as delay and loss rates relating to traffic flows are collected at intermediate nodes. A control server processes the collected data to determines data flow rates for different priorities of traffic. A static directory node is used to look up inter-node connections and determine initial traffic classes corresponding to those connections. The rates are combined with the initial traffic classes to define codes for encoding the headers of packets to determine their network priority.

18 Claims, 3 Drawing Sheets



SLA SCENARIO OF SEPARATE DIRECTORY & CONTROL SERVERS

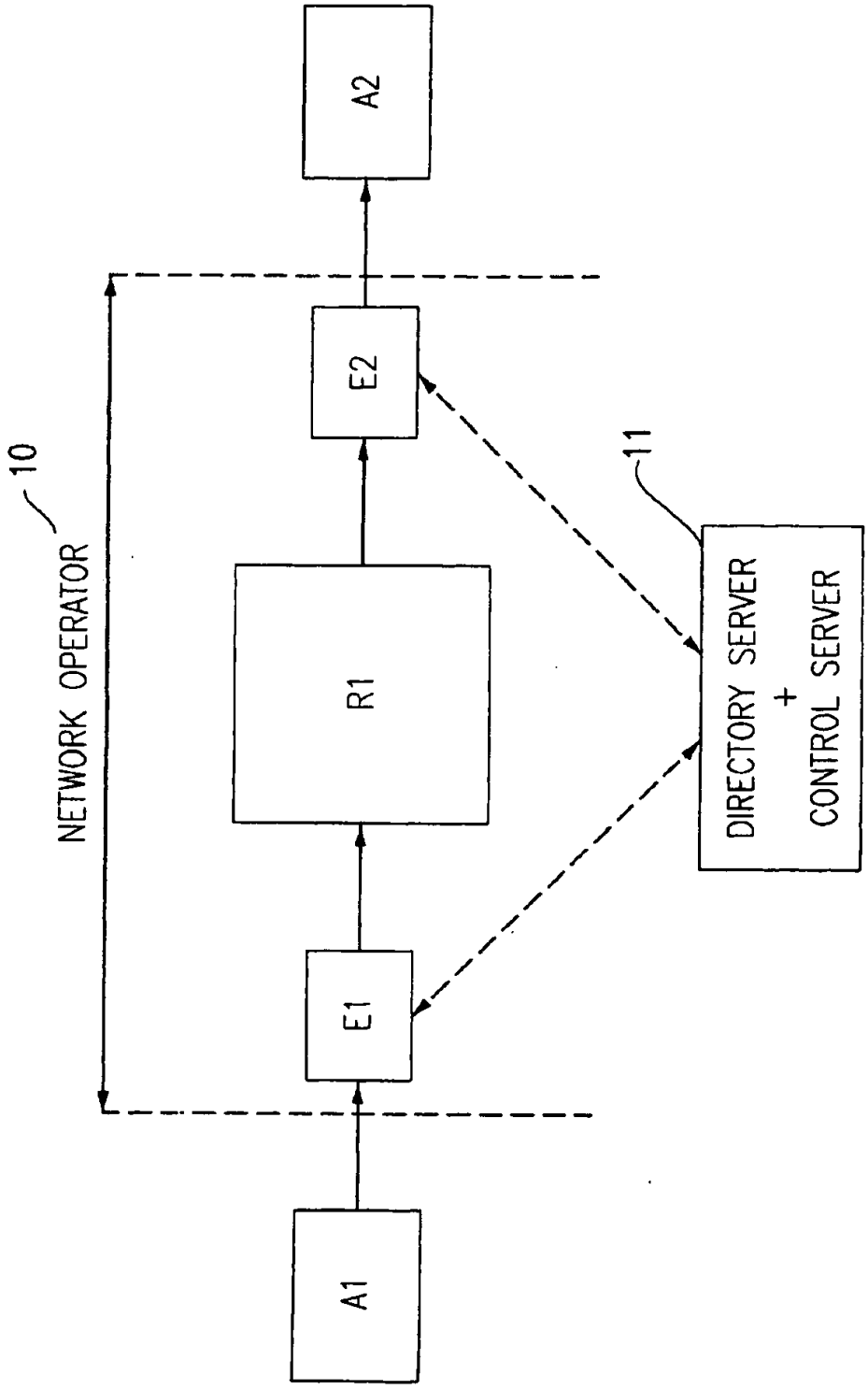


FIG.1A

SLA SCENARIO OF INTEGRATED DIRECTORY & CONTROL SERVER

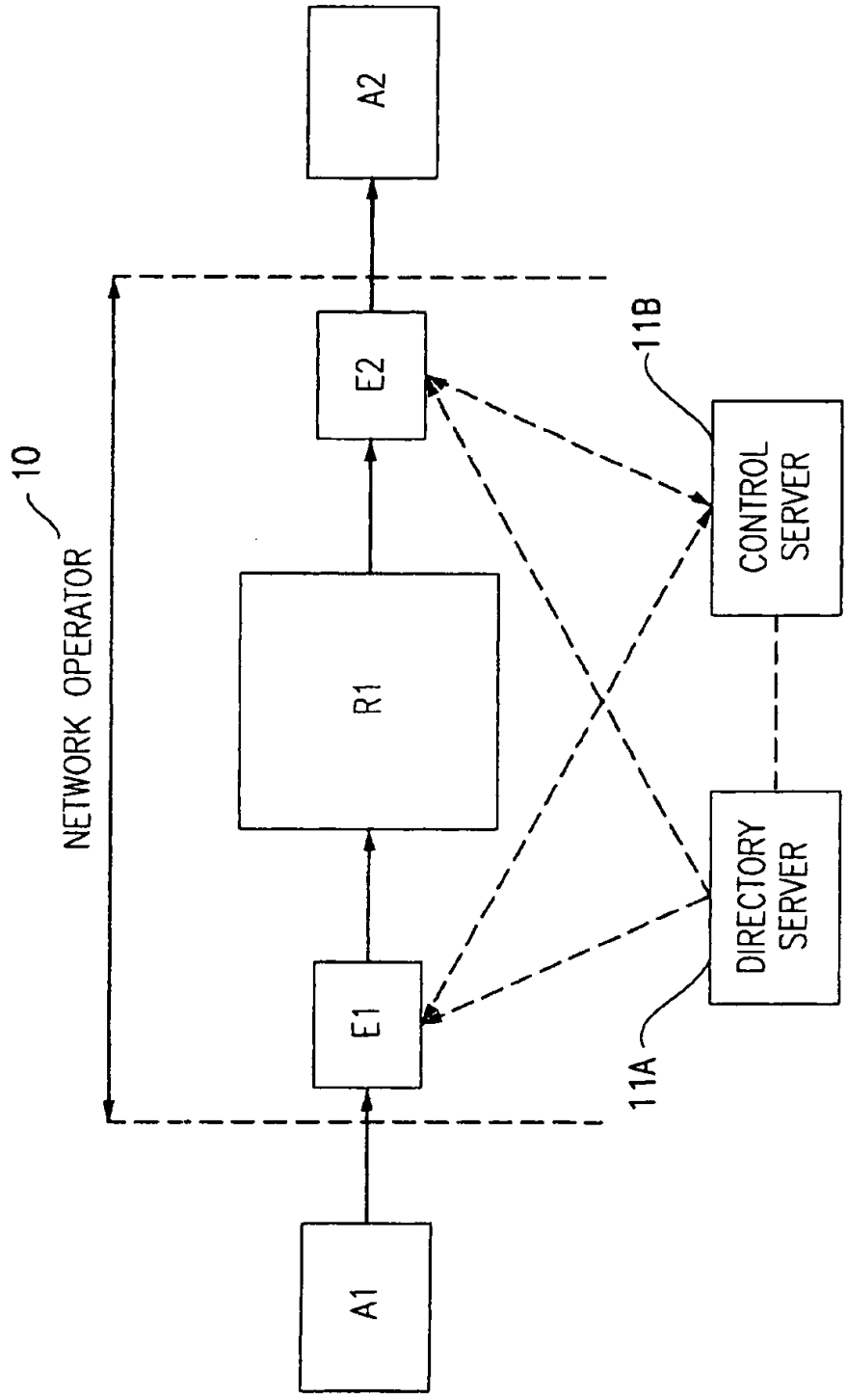
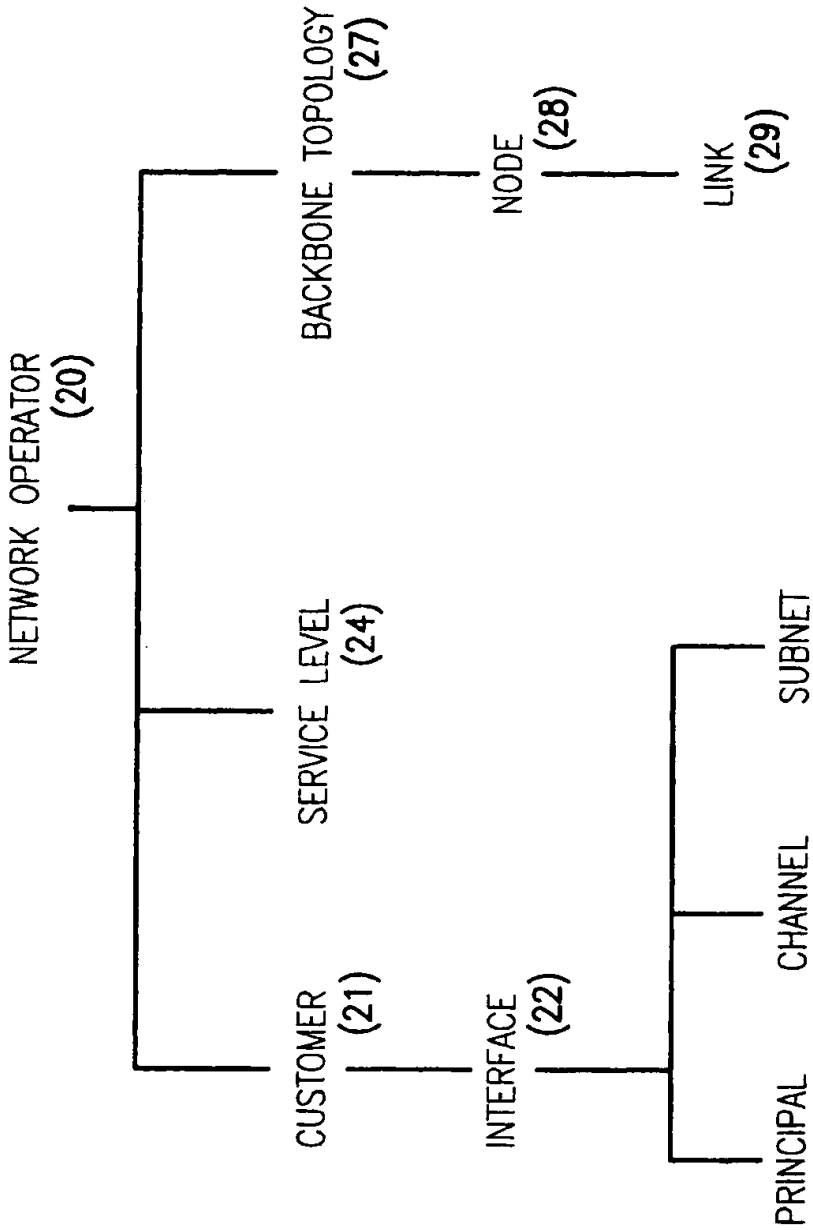


FIG. 1B
SLA SCENARIO OF SEPARATE DIRECTORY & CONTROL SERVERS



DIRECTORY TREE STRUCTURE

FIG.2

ARCHITECTURE FOR SUPPORTING SERVICE LEVEL AGREEMENTS IN AN IP NETWORK

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to architectures for delivering networking products across multiple platforms, and in particular to architectures for delivering Internet Protocol (IP) networking products which are enabled to support service level agreements.

2. Background Description

The operators and users of enterprise networks prefer that their networks be predictable and provide consistent performance. Predictability and consistency are often more important than the raw capabilities of the network, i.e. a network that provides a consistent medium throughput is often considered more desirable than a network which provides very high throughput at some times, but performs poorly at other times. For many business applications, it is important that transactions be completed in a predictable manner while the time taken for the transactions to complete is relatively unimportant (provided it does not exceed a reasonable limit).

Prior art solutions, such as SNA, provide network predictability by preconfiguring the network. This does not work in an IP network, because IP is dynamic and connectionless, and therefore relatively unpredictable. The typical enterprise network environment consists of several campus area networks interconnected by a wide area backbone network. The campus networks usually deploy high-speed links, and perform reasonably well. Congestion tends to occur in the backbone network, which consists of relatively slower speed point-to-point links, and in some of the campus networks which house the servers.

An approach is needed which will provide predictability on an IP backbone network, and do so for backbones with varying degrees of capability. If the network provider can predict the performance of the network, then he can implement service level agreements. A service level agreement is a formal contract entered into by a service provider and its customers. The service provider contracts to transport packets of electronic data between customer premise networks (branch offices, data centers, server farms, etc.) across the provider's backbone network with certain assurances on the quality of the transport. This is known as the Service Level Agreement (SLA). The SLA specifies customer expectations of performance in terms of parameters such as availability (bound on downtime), delay, loss, priority and bandwidth for specific traffic characteristics. An SLA includes acceptable levels of performance, which may be expressed in terms of response time, throughput, availability (such as 95% or 99% or 99.9%), and expected time to repair.

SLAs vary greatly from one network to the next, and from one application to another running on the same network. They are normally based on some level of expected activity. For example, if a large airline wants to ensure that the lines at the ticket counter do not get overly long due to poor response time at the ticketing terminals, some estimate must be made of expected workload, so that the network administrator can be prepared with the necessary resources to meet that workload and still remain compliant with the performance terms of the SLA.

Managing an SLA is an important task because of the revenue implications of failure to support mission-critical business applications. The problem is exacerbated due to diversity of the traffic and due to poor and varying degree of service differentiation mechanisms within the backbone networks. Commercially significant traffic must be prioritized above workloads which do not have a critical time dependency for the success of the business. Many of these workloads in an IP environment are far more volatile than those which have traditionally been encountered in the prior art, e.g. in native SNA environments. In order to meet customer requirements in this environment, a service provider must provide a large excess capacity at correspondingly high charges.

This situation dramatizes the need for effective tools which can monitor the performance of the IP network or system delivering a service over the IP network. While SLA management tools already exist in the native SNA VTAM environment, these tools do not generally exist for IP backbones. Also, there is a need for effective controls which allow the service provider of an IP network to manipulate the priority of the various workloads to be managed.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide support for service-level agreements in a corporate itinerant or an ISP-controlled portion of the Internet.

It is also an object of the invention to provide tools which can monitor the performance of an IP network as measured against multiple SLA agreements.

It is a further object of the invention to provide effective controls which allow the service provider to manipulate the priority of the various workloads subject to SLA agreements.

Another object of the invention is to provide means for achieving network predictability which are adequate to implement a variety of SLA agreements over IP backbone networks halving a variety of capabilities.

It is yet another object of the invention to provide network traffic control tools enabling optimum allocation of network resources and minimizing the need to provide excess capacity in order to implement a variety of SLA agreements.

This invention discloses an architecture (SLA architecture) which organizes the key components, the specific function placements and communication mechanisms so as to enable efficient means of implementing new tools which greatly facilitate both development and enforcement of an SLA. Further, these advantages are even more significant when the backbone network such as current IP-based networks provide very little means for such service differentiation.

The key components of a service level agreement are availability and responsiveness. Availability is maintained by managing network connectivity in the presence of failures, and responsiveness by maintaining a satisfactory Level of network performance. In an IP network, availability is largely taken care of by the adaptive routing mechanism used by IP, but responsiveness needs to be managed. The schemes that make the network predictable provide mechanisms that can estimate the responsiveness of an IP network, and thereby assist in implementing service level agreements. The approach taken in accordance with the present invention to provide predictability in an IP network is to provide a quasi-static configuration which adapts to longer term fluctuations of traffic and relies upon the dynamism of IP to react properly to short term fluctuations and congestion.

Quasi-static adaptations may be viewed as dynamic in relation to longer time scales. By extending the adaptive

time scales to relatively gross periods of hours, days and weeks, as appropriate, a quasi-static configuration enables the network to modify allocation of resources in such a manner as to lower the load on the network, in contrast to prior art techniques such as Response Reservation Protocol (RSVP) which allow necessary resources to be requested bit impose a higher signalling load on the network.

The invention involves controlling packet traffic in an IP network of originating, receiving and intermediate nodes to meet performance objectives established by service level agreements. To implement the invention, traffic statistics and performance data such as delay and loss rates relating to traffic flows are collected at intermediate nodes. A central server processes the collected data to determines rates for different priorities of traffic. A static directory node is used to look up inter-node connections and determine initial traffic classes corresponding to those connections. The rates are combined with the initial traffic classes to define codes for encoding the headers of packets to determine their network priority.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

FIG. 1*a* is a block diagram of a service level agreement scenario having a combined directory server and control server; FIG. 1*b* shows the same scenario having a directory server distinct from the control server.

FIG. 2 is an illustration of a hierarchical structure of a directory that can be used for Service Level Agreements.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT OF THE INVENTION

Referring now to the drawings, and more particularly to FIG. 1*a*, there is shown a Service Level Agreement scenario, with a Network Operator **10**, Customer Premise Networks A1 and A2, Edge Devices E1 and E2, a Network Router R1 and a Directory Server/Control Server **11**. FIG. 1*b* shows the same scenario, but with Directory Server **11**A distinct from Control Server **11**B.

The main components of the proposed architecture are:

Edge Device(s),

Control Server(s),

Directory Server(s),

Edge Device to Control Server Protocol,

Edge Device to Directory Server Protocol,

Control Server to Directory Server Protocol, and

End Host Protocol.

These will now be discussed in detail.

An Edge Device in the SLA architecture is a module that interfaces a customer premise network with the backbone network. (Currently, backbone networks vary widely in their resource management and service differentiation capabilities (e.g. an IP network with support for resource reservation and/or support for differential services using weighted fair queuing (WFQ) or class based queuing (CBQ), an ATM or Frame Relay network supporting switched virtual circuits with committed rates, etc). Such heterogeneity is expected to continue as vendors of networking equipment seek to differentiate their products. In such an environment, edge devices play the role of adapting the traffic entering the backbone network to the specific capabilities provided by the network in order to ensure that the SLA conditions are met efficiently.

An Edge Device may reside on a stand-alone processing device or be integrated into the border router that acts as a gateway to the service provider network. In either case, all packets originating in one customer premise network and destined for another pass through two Edge Devices components; i.e., the ingress Edge Device E1 at the interface between the backbone network and the source customer premise network A1, and the egress Edge Device E2 at the interface between the backbone network and the destination customer premise network A2. Note that customer premise networks A1 and A2 are not transit networks.

The ingress Edge Device E1 obtains some or all of the following information, either carried in the packets received from the customer premise network, or obtained through a lookup based on information stored at the edge device:

ingress interface,

source address,

source port,

destination address,

destination port,

protocol id,

Class of Service identification,

contents of packet,

header fields in transport protocol.

An ingress Edge Device E1 performs some or all of the following operations on packets that it intercepts as they leave a customer premise network A1 and enter the backbone network.

1. Classification: Packets are categorized into separate streams based on a number of criteria that depend on the terms of SLA and the network capabilities. The Edge Device uses a set of classification rules to determine the appropriate service level category to which the packet is assigned. These rules may be configured in the Edge Device or obtained by querying a Directory Server. The details of the latter mode of operation will be discussed below in the context of the Directory Server operation. In a preferred implementation, only the egress edge device classification and class of service classification are necessary to provide service level agreements. For finer granularity control, the other classifications (path, channel, flow) can also be used.

(a) Egress Edge Device Classification: The ingress Edge Device E1 that receives the packet from the customer premise network A1 obtains the identity of the remote or egress Edge Device E2 that the packet is expected to traverse before being delivered to the destination customer premise network A2, either directly from the packet or based on a lookup.

(b) Path Classification: The ingress Edge Device E1 determines the path that is expected to be traversed across the backbone network by the packet.

(c) Class of Service (classification: Packets with similar designated service categories are considered to belong to same stream. The class of service may be determined directly from information carried in the packet or may be based on other header fields carried in the packet, or based on a set of classification rules at the Edge Device.

(d) Channel classification: A channel is defined as a stream of packets that have the same ingress and egress edge devices, that are expected to follow the same path through the network and have the same Class of Service. The present invention also covers the case where all packets expected to traverse the same remote

edge device are classified into the same channel, irrespective of the expected path within the network.

(e) Flow classification: A flow is the basic packet stream unit over which a service level agreement may be specified. Typically, all packets in a flow belong to the same channel.

2. Packet Transformation: The edge-device is responsible for changing packet formats so that the backbone network is capable of mapping the transformed packet into the right class for purposes of scheduling and buffer management. Two common forms of transformation are 1) to change the contents of the TOS (Type of Service; field in the IP header; or 2) to encapsulate the packet with an additional IP header. When the backbone network supports a mechanism such as WFQ or CBQ which operates on the TOS byte field, the edge-device changes the contents of the TOS byte to a value specific to the class of service assigned to a packet. When the backbone network supports a mechanism such as RSVP based reservation, or WFQ/CBQ which operate on the basis of port numbers in the transport header, an encapsulation into an additional IP header is required. The external IP header would contain the right fields which would permit the routers in the backbone network to classify it easily.

Packet transformation may need to be done both at the entry and the egress edge-device for a packet. When only the TOS byte value is changed, only the entry edge-device needs to transform the packet. However, when an additional IP header is used for encapsulation, the entry edge-device transforms the packet by adding the external IP header, and the egress edge-device transforms the packet by removing the external IP header and restoring the original packet.

3. Scheduling: Scheduling refers to the differential treatment given to different flows in terms of access to link bandwidth. Typically, each outgoing flow at the edge de(vice is assured of the opportunity to share available link bandwidth fairly with other contending flows. In this context, it becomes important for Edge Devices E1 or E2 to schedule packets awaiting acess to the link, sending them out in a certain order, perhaps different from the order in which the packets were received. Scheduling may also aggregate similar flows and arbitrate amongst the aggregates.

4. Buffer management: Another resource at the edge device that needs to be shared is buffer space. Buffer management refers to the operations of the edge device to assure a fair treatment of different flows in terms of their access to this resource, based on their priority and current usage of buffers.

5. Policing: The SLA specifies the service levels that individual flows should receive as long as the traffic generated by these flows is within specified bounds. The policing functionality checks for violation of the traffic contract by flows and may penalize certain applications by degrading their service level temporarily (marking/dropping all such packets).

6. Pacing: During congestion states within the network, certain channels may be affected because they use congested portions of the network. As will be discussed later, the Control Server component of the SLA architecture is capable of detecting both the congestion state as well as affected flows. Under the directive of the control server, an Edge Device will regulate the rates of affected active channels to alleviate the impact of congestion.

7. Statistics collection: An Edge Device maintains various counters to monitor the traffic rates of flows in either direction.

8. Traffic prediction: This involves using the collected statistics to forecast near-term traffic (and the consequent resources requirement) of flows that will enter the backbones network from the Edge Device.

9. Performance monitoring: This includes estimating the bandwidth, delay and loss characteristics of selected flows. This function will be realized either using additional probe packets or using header fields if data packets are encapsulated before entering the backbones network. The frequency of probing is adjusted according to the SLA terms while maintaining a tight control over the overhead introduced by such additional traffic. The latter is achieved by ensuring that the overhead of probing does not exceed a certain percentage of the actual data traffic which is monitored by the statistics collection function. Performance monitoring can be done at the egress edge device only, or at a combination of the ingress and egress edge devices.

10. Policy control: This covers a variety of operations performed at the edge of the network, including access control, security, billing, etc. Network administrators may wish to allow or disallow the use of network resource based on the origin, destination or protocol used by the packet stream. In addition, policy control may involve authentication of applications and their desired service levels in an environment where end-hosts are capable of signaling their resource requirements/service prior-ities directly to the network. This function involves communication with a directory/policy server described below.

2. Control Server

A control server in the SLA architecture is a module that acts as a repository of dynamic information (in accordance with the above referenced "quasi-static" approach involving adaptive time scales), e.g. resource utilization within a portion of the backbone network. Based on the knowledge of the topology, resource utilization and service level agreements with all customer premise networks, the control server computes the allocation of backbone network resources, and informs the edge devices of the pacing that must be done on various channels. To this end, the control server may perform some or all of the following functions:

1. Network Topology Learning: The control server becomes aware of the topology and total resource availability at network elements. These network elements may include various edge devices, routers, switches, bridges or links between other network elements. The resources at the network elements may include bandwidth, packet processing speeds, buffers, and protocols to manage these resources. The control server can obtain this information directly, i.e., through participation in routing protocols, or network management protocols or through configuration; or indirectly, i.e. from the Directory server or edge devices.

2. Updating Directory server: If the Control Server obtains the topology and network resource information directly then it updates the Directory Server accordingly.

3. Network Device configuration: In the event that the resource management protocols at various network devices are capable of remote configuration then the control server may take on the task of configuring them accordingly. In particular, the control server may adjust the parameters of the link bandwidth scheduling protocols at various routers. These adjustments will be propagated automatically, without the need for rebooting the devices thereby reconfigured.

4. Determining routing topology: Periodically, the control server obtains routing tables and other routing information relevant, to the routers and edge devices, in order to remain aware of the latest routing topology of the backbone network. This may be done through participation in routing protocols or polling routers/edge-devices through network management protocols.

5. Polling Edge Devices for Channel Statistics: Periodically, the control server polls the edge devices, and obtains the current set of active channels, the bandwidth utilization of these channels, other traffic statistics that are relevant to network resource use, as well as any estimates of future resource use computed by the edge devices.

6. Load balancing: If the control server is capable of determining the routing in the network, then it uses this capacity to balance the load in the network.

7. Congestion avoidance: The control server may detect that certain network devices, such as routers and links, are overloaded. In this case the control server will compute the channels that are over-using their resource allocations and inform the appropriate ingress edge devices corresponding to these channels, advising them to pace the channels.

3. Directory Server

The directory server is responsible for maintaining information which is relatively static. It is organized as per the specifications of The Directory: Overview of Concepts, Models and Service, CCITT Recommendation X.500, 1988. Information in the directory server is used to maintain information about classifying packets into one or more service-levels, to maintain policy information regarding applications, and to maintain information about the different service-levels that is to be expected of different customers.

The directory server is represented as an X.500 directory and is accessible via the LDAP protocol (*Lightweight Directory Access Protocol RFC* 1777, March 1995, by W. Yeong, T. Howes and S. Kille). It maintains three main types of information:

- Information about the policy rules for applications and users, mapping traffic to one of the several service levels of the class.
- Information about the mapping of service-levels to the appropriate set of performance requirements.
- Information about the topology of the network, and the characteristics of the communication channels in the network.

All the entries relevant to Service Level Agreements are rooted at the entry identifying the Network Operator **10**. A hierarchical structure of the directory that can be used for Service Level Agreements is illustrated in FIG. **2**.

Each entry in the directory belongs to one of the categories shown above, and is assigned a relative distinguished name at each step of the path it belongs to.

The category Network Operator **20** identifies the organization that is responsible for this portion of the network. It contains all the standard attributes associated with an organization in a standard X.500 directory.

The Customer category **21** identifies different customers supported by the network operator. Each customer has all the attributes associated with an organization or an individual. Additionally, it contains a list of all the interfaces associated on which the customer packets can be received. These interfaces are identified by their unique IP addresses within the operator network.

The Interface category **22** identifies an interface through which a customer may send its traffic. The access to an interface might be via dial-up lines or via a directly connected network. An interface is identified by its IP address, and has a default service level which is assigned to its owners. It also contains the name of the owner and the physical machine on which it is installed. An interface entry also contains the time when it was last updated.

The Principle category **23** identifies the type of rules that are used to determine the Service Level **24** to which traffic should be assigned. Each principle is a subset of the traffic that will be assigned to a specific interface. Each entry in this category has a type attribute which determines whether the principle can be identified by looking at the Type of Service (TOS) byte in an incoming IP packet, whether it can be identified by looking at IP header information only, whether it can be identified on the basis of IP and TCP/UDP headers, or if the principle is identified by means of an URL or by other upper layer information. It identifies the interface to which the principle is applicable. It may be applicable to more than one interface. A special interface address (IP addr 0.0.0.0) is used to identify a principle that is applicable to all the interfaces belonging to a particular customer.

The Service Level attribute **24** identifies the service level to which the traffic matching the specifications of the principle could be mapped onto.

The specification of the principle is identified by the type of principle. If the principle type is interface only, no further specification is needed. All packets coming on the specified interface are mapped to the specified service-level. If the principle type is IP header only, it may contain the TOS byte of the incoming packet which defines the principle. All packets on the interface with specified TOS byte value are mapped onto the specified service level. If the principle type contains IP and TCP/UDP headier, the additional attributes would be the source-destination IP addresses and the TCP/UDP port numbers which would need to be specified.

The Channel category **25** identifies the different logical connections that are supported by the network operator for a particular customer. The channel contains the source and destination interface addresses, as well as the required performance and traffic level that should be supported on the channel. Other attributes of the channel can also be stored in this entry. Both the desired and the actual observed performance of the channel is stored in this entry.

The Subnet category **26** identifies the topology behind the interface. The subnet is characterized by an IP address and a subnet mask. Multiple subnets that belong to the customer, and are accessible through the interface, are enumerated here.

The Service Level category **24** identifies the different service-levels that are offered by the network operator. An entry in this category would contain the service-level name, and the method used to encode the service in the backbone network. The method used may specify the TOS byte to be used by packets, or a specific port and protocol within which such traffic should be encoded. It also contains the performance specifics required for service-level, including the round-trip delay or loss-rate. The performance specifics are specified as an incremental add-on to the basic performance that is determined by the properties of the source and destination address for a specific communication channel. The entry also specifies the action to be taken when traffic belonging to this service-level is found to be in violation of an assigned traffic rate. The action could take the form of dropping packets, buffering packets, downgrading packets to a different service-level, or downgrading sessions to a different service-level. In the latter case, the next service level also needs to be specified.

The Backbone Topology class **27** is used to store the topology of the network and the network state. Below its hierarchy, one would find the different nodes **28** and links **29** which will constitute the backbone network, and their state, including their present utilization, their capacity etc. In some implementations, it may not be necessary to support the full directory tree structure shown in FIG. **2**. For example, the backbone topology need not be stored in cases where the control server determines backbone topology dynamically.

Similarly, if the network is dedicated to a single customer, the customer class 21 can be eliminated.

4 Edge device—Directory Server Protocol

The system can operate in one of two modes. In the first mode, the directory server and the control server are integrated, and are accessed using a directory access protocol such as LDAP. In the second mode, the directory server and the control server are different entities (which may be co-located on the same machine), each supporting a different set of protocols. The directory server would always use a protocol such as LDAP, although the control server is likely to use a different protocol to counter the difficulties associated with managing highly dynamic data using LDAP.

In both of the modes, the edge-device registers itself with the Directory Server when it is initiated. The edge-device Would identify the interface address to which it is attached, and registers the subnets behind it with the directory server. It would also obtain the list of other interfaces belonging to the same customer, and the subnets associated with them from the directory server. The edge-device would then query the directory server for the different principles that are defined for it. These principles provide the rules that map the traffic on the network from the edge-device to the different levels of service in the backbone network. The set of channels belonging to the interface are also queried by the edge-device. These provide the values for the different traffic rates associated with each channel. Any other policy information is also obtained by querying the directory server.

When the directory server is integrated with the control server, the edge-device would periodically query the directory for the different channels for which an interface on the edge-device is an end-point. For these channels, the edge-device would obtain the specified maximum traffic to be placed in the network, as well as the specific performance expected of the channel. The edge-device only sends out packets according to these specifications into the network. The edge-devices would also monitor the performance of the packets, and update the status of the channel in the directory as to whether the performance specs are being met or not.

5. Edge Device—Control Server Protocol

When the control server and directory server are integrated, the protocol used between the two is as defined in the previous section. However, when the control sever is a different entity, a different polling protocol is used by it to monitor the state of the network.

When it is initiated, the edge-device is required to register with the control server. The control server, at periodic intervals, polls each of the edge-devices to obtain the performance and traffic characteristics of the channels belonging to each edge-device. The control server would use this information to determine which edge-devices, if any, should be regulated, and allocate the pacing rate to the ones being regulated. The control server uses an adaptive dynamic algorithm to determine the rates, and the set of channels which need to be paced. A set of channels may need to be paced together since they all share the same bottleneck link. This mapping information is also sent by the control server.

The rates to be assigned to different channels can be computed by the following adaptive dynamic algorithm, where the network state is collected at regular periods and the algorithm is invoked at the end of each period. The algorithm seeks to determine the maximum utilization of a network node/link which would satisfy the performance requirements of all channels:

1. For all the channels in the network, determine the following quantities: a) mean actual traffic observed in the previous period; b) forecasted mean traffic expected for the next period; c) whether the channel's performance requirements were satisfied in the last period.

2. On the basis of the foregoing collected information, determine the utilization of each node and link in the network for the last period. The utilization is given as the ratio of actual traffic in the last period to the traffic capacity of the node/link.

3. For all channels whose performance was not met in the previous period, identify the node/link with the highest utilization long its path. For this path, revise the maximum utilization of the node/link to be the lower of the current maximum utilization and the actual utilization.

4. If all channels through a node/link have met their performance in the last period, revise the maximum utilization of the node/link to be the higher of current maximum utilization and the actual utilization.

5. For each channel for which the maximum utilization of each node/link along its path is larger than the predicted utilization for the next period, make its assigned rate be the same as the predicted rate.

6. For all nodes/links for which the predicted utilization in the network exceeds the maximum utilization in the network, repeat steps 7 through 8.

7. Assign the assigned rate to the channels passing through the node by the ratio of the maximum utilization to the actual utilization.

8. Recompute the predicted utilization of the nodes/links in the network.

6. End Host Protocol

In the normal mode of operation, the end host would not be expected to participate in any interaction with the directory server or the control server. However, new applications which are aware of the directory server can use a protocol to query the directory and obtain their service-level information. Some of the applications may be capable of marking their service-levels in the end-host, and the edge-device function would be to verify that the end-host marking is consistent with the schema as defined by the directory. In some cases, the end host may update the entries in the directory which would be queried by the edge-device to obtain the most current rules.

While the invention has been described in terms of a single preferred embodiment, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is as follows:

1. A method of controlling packet traffic in an IP network, comprising the steps of:

at one or more edge-nodes in a connectionless network, identifying inter-node connections and determining their corresponding initial traffic classes and traffic flows by looking up said initial traffic classes in a relatively static directory server, said relatively static directory server having a configuration that adapts to longer term fluctuations of traffic;

transforming packets belonging to said inter-node connections to encode information about said traffic classes; and

for one or more of said traffic flows, regulating the transmission rate of packets belonging to each of said one or more traffic flows to meet performance objectives according to service level agreements.

2. The method of claim 1, wherein said regulation of transmission rate includes setting an upper limit for said rate.

3. A method of controlling packet traffic in an IP network, comprising the steps of:

at one or more edge-nodes in a connectionless network, identifying inter-node connections and determining their corresponding initial traffic classes and traffic flows;

transforming packets belonging to said inter-node connections to encode information about said traffic classes; and

for one or more of said traffic flows, regulating the transmission rate of packets belonging to each of said one or more traffic flows to meet performance objectives according to service level agreements,

wherein said regulation of transmission rate includes setting an upper limit for said rate, and

wherein said regulation of packet transmission rate includes determining said rate by the further steps of:

at edge-nodes, collecting traffic statistics and performance data relating to a traffic flow;

processing the information from said collecting step to determine data flow rates for different priorities of traffic;

repeating said collecting and processing steps at periodic intervals, and propagating said rate to individual nodes on said network in a quasi-static mode.

4. The method of claim **3**, wherein said regulation of rates is done by a control server which dynamically tracks traffic in the network and determines rates for traffic flows at one or more network nodes, said network nodes being either edge-nodes or intermediate nodes.

5. The method of claim **4**, wherein said control server configures devices on the network to efficiently support the rates for traffic flows at network nodes, said configuration being done when new rates are computed at the control server.

6. The method of claim **3**, wherein said regulation of packet transmission rate includes determining said rate by storing and querying entries in a directory server.

7. A method of controlling packet traffic in an IP network, comprising the steps of:

at one or more edge-nodes in a connectionless network, identifying inter-node connections and determining their corresponding initial traffic classes and traffic flows;

transforming packets belonging to said inter-node connections to encode information about said traffic classes; and

for one or more of said traffic flows, regulating the transmission rate of packets belonging to each of said one or more traffic flows to meet performance objectives according to service level agreements,

wherein said regulation of transmission rate includes setting an upper limit for said rate, and

wherein said initial traffic classes and said encoding are changed by edge-nodes to form new traffic classes and new encoding for said inter-node connections.

8. The method of claim **7**, wherein for each of one or more of said inter-node connections said new traffic class is of a lower priority than said initial traffic class, and said change to said new traffic class is made when said inter-node connection is sending packets in the network at a rate beyond said upper limit.

9. An apparatus for controlling packet traffic in an IP network, comprising:

at one or more edge-nodes in a connectionless network, means for identifying inter-node connections and determining their corresponding initial traffic classes and traffic flows, said means for determining said initial traffic classes further comprising means for looking up said initial traffic classes in a relatively static directory server, said relatively static directory server having a configuration that adapts to longer term fluctuations of traffic;

means for transforming packets belonging to said inter-node connections to encode information about said traffic classes; and

for one or more of said traffic flows, means for regulating the transmission rate of packets belonging to each of said one or more traffic flows to meet performance objectives according to service level agreements.

10. An apparatus as in claim **9**, wherein said means for regulating the packet transmission rate includes means for setting an upper limit for said rate.

11. An apparatus as in claim **10**, wherein said regulating means further comprises:

means for collecting statistics about traffic flows; and

means for collecting performance information about traffic flows.

12. An apparatus as in claim **10**, further comprising means for changing the initial traffic class of an inter-node connection to a new traffic class.

13. An apparatus as in claim **11**, wherein said regulating means further comprises means for determining if the service level agreements of a traffic flow are being satisfied.

14. An apparatus as in claim **11**, wherein said rates for said traffic flows are regulated according to said traffic statistics and said performance information.

15. The method of claim **9**, wherein said identifying and transforming means are implemented at edge devices.

16. An apparatus for computing the rates for traffic flows at individual nodes in an IP network, comprising:

means for determining the routing topology of a connectionless network;

means for collecting statistics about traffic flows in the network;

means for collecting performance information about traffic flows in the network; and

means for combining said routing topology with said statistics and said performance information to determine the rates for traffic flows in the network to meet service level agreements,

wherein said combining means is accomplished at periodic intervals and said rates are disseminated to network nodes in a quasi-static mode.

17. An apparatus as in claim **16**, further comprising means for configuring network devices to optimally support said traffic rates.

18. A method for optimizing resource utilization among customers of an IP network, comprising the steps of:

defining service level agreements for each said customer;

establishing a control server as a dynamic repository of network information, said information including resource utilization, topology, and service level agreements;

receiving said topology information at said control server, said topology information including edge devices through which said customers connect to the network;

establishing a directory server as a quasi-static repository of network information, said information including policy rules for mapping traffic to service levels, and for mapping service levels to performance requirements;

monitoring traffic on said network at each of a plurality of edge devices, said edge devices operating to classify said traffic;

using said control server to compute the allocation of backbone network resources and issue pacing instructions to said edge devices; and

propagating directory server information to network devices automatically and without reconfirming the network, said propagation being accomplished dynamically over long time scales,

wherein said network is connectionless.

* * * * *

TOS Field

US006651101B1

(12) **United States Patent**
Gai et al.

(10) **Patent No.: US 6,651,101 B1**
(45) **Date of Patent: *Nov. 18, 2003**

(54) **METHOD AND APPARATUS FOR IDENTIFYING NETWORK DATA TRAFFIC FLOWS AND FOR APPLYING QUALITY OF SERVICE TREATMENTS TO THE FLOWS**

(75) Inventors: **Silvano Gai**, San Jose, CA (US); **Keith McCloghrie**, Middletown, NY (US); **Shai Mohaban**, Sunnyvale, CA (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/191,605**

(22) Filed: **Jul. 9, 2002**

Related U.S. Application Data

(63) Continuation of application No. 09/911,122, filed on Jul. 23, 2001, now Pat. No. 6,434,624, which is a continuation of application No. 09/206,067, filed on Dec. 4, 1998, now Pat. No. 6,286,052.

(51) **Int. Cl.**[7] **G06F 15/16**

(52) **U.S. Cl.** **709/224**; 709/235; 709/238

(58) **Field of Search** 709/223, 224, 709/225, 238, 249, 311; 370/235

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,769,810	A	9/1988	Eckberg, Jr. et al.
4,769,811	A	9/1988	Eckberg, Jr. et al.
5,224,099	A	6/1993	Corbalis et al.
5,263,157	A	11/1993	Janis
5,473,599	A	12/1995	Li et al.
5,606,668	A	2/1997	Shwed

(List continued on next page.)

OTHER PUBLICATIONS

Ortiz, Jr. Sixto, Active Networks: The Programmable Pipeline, Computer, August 1998, pp. 19–21.

IEEE 0802.1D/D15, Standard (draft 15) " Information Technology–Telecommunications and information exchange between systems—Local and Metropolitan Area Networks—Common Specifications—Part 3: Media Access Control (MAC) Bridges Revision" Nov. 24, 1997, pp. 1, 50–56, 378–381.

An Emerging Trend in the Internet Services Market, http://www.hpismanagment.com/fh/products/prowhtppr.htm, pp. 1–10.

Wroclawaski, J., "The Use of RSVP with IETF Intergrated Services", RFC 2210 Network Working Group of IETF, Sep. 1997, pp. 1–27.

Bernet, Y. et al., "A Framework for Use of RSVP with Diff-serv Networks", IETF Internet Draft, Nov. 1998, pp. 1–21.

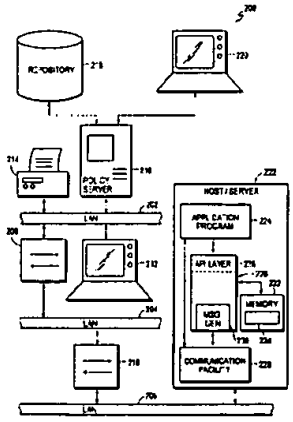
(List continued on next page.)

Primary Examiner—Viet D. Vu
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP; Michael R. Reinemann

(57) **ABSTRACT**

A system within a computer network identifies specific traffic flows originating from a given network entity and requests and applies appropriate policy rules or service treatments to the traffic flows. A network entity includes a flow declaration component that communicates with one or more application programs executing on the entity. The flow declaration component includes a message generator and an associated memory for storing one or more traffic flow data structures. For a given traffic flow, the application program issues one or more calls to the flow declaration component providing it with information identifying the traffic flows. The flow declaration component then opens a flow management session with a local policy enforcer that obtains policy rules or service treatments for the identified flow from a policy server and applies those rules or treatments to the specific traffic flows from the network entity.

16 Claims, 10 Drawing Sheets



U.S. PATENT DOCUMENTS

5,666,353	A		9/1997	Klausmeier et al.
5,751,967	A		5/1998	Raab et al.
5,819,042	A		10/1998	Hansen
5,832,503	A		11/1998	Malik et al.
5,842,040	A		11/1998	Hughes et al.
5,872,928	A		2/1999	Lewis et al.
5,889,953	A		3/1999	Thebaut et al.
5,987,513	A		11/1999	Prithviraj et al.
6,021,263	A		2/2000	Kujoory et al.
6,041,347	A		3/2000	Harsham et al.
6,046,980	A		4/2000	Packer
6,047,322	A		4/2000	Vaid et al.
6,078,953	A		6/2000	Vaid et al.
6,091,709	A		7/2000	Harrison et al.
6,098,099	A		8/2000	Ellesson et al.
6,101,541	A		8/2000	Ellesson et al.
6,104,700	A		8/2000	Haddock et al.
6,157,955	A	*	12/2000	Narad et al. 709/228
6,181,694	B1	*	1/2001	Pickett 370/353
6,308,216	B1		10/2001	Goldsmidt et al.

OTHER PUBLICATIONS

Bernet, Y. et al., "Requirements of Diff-Serv Boundeary Routers", IETF Internet Draft, Nov. 1998, pp. 1–30.

Yadav, S. et al., "Identity Representation for RSVP", IETF Internet Draft, Jan. 1999.

Heinanen, J. et al., "Assured Forwarding PHB Group", IETF Internet Draft, Sep. 1998., pp. 1–6.

Jacobson, V. et al., "An Expedited Forwarding PHB", IETF Differentiated Services Working Group, Aug. 1998, pp. 1–7.

Nichols, K. et al., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and Ipv6 Headers", IETF Differentiated Services Working Group, Internet DRaft, Aug. 1998, pp. 1–18.

Blake, S. et al., "An Architecture for Differentiated Services", IETF Differentiated Services Working Group, Internet Draft, Aug. 1998, pp. 1–30.

Bernet, Y et al., "A Framework for End-to-End Qos Combining RSVP/Interserv and Differentiated Services", IETF, Internet Draft, Mar. 1998, pp. 1–13.

Yavatkar, R. et al., "A Framework for Policy-based Admission Control", IETF, Internet Draft, Nov. 1997, pp. 1–21.

Boyle, J. et al., "The COPS (Common Open Policy Service) Protocol", IETF, Internet Draft, Aug. 6, 1998, pp. 1–23.

Reichmeyer, F. et al., "COPS Usage for Differentiated Services" IETF, Network Working Group, Internet Draft, Aug. 6, 1998, pp. 1–16.

"Cisco IOS Software Quality of Service Solutions", Cisco Systems, Inc., Jul. 1998, pp. 1–29.

"Queueing, Traffic Shaping, and Filtering", Cisco Systems, Inc., (c) 1992–1999, pp. 1–2.

"Network Node Registry Overview", http://www.net.gov.bc.ca/NNR/nnrintro.htm., Jan. 29, 1998, pp. 1–3.

"Network Node Registry User's Guide", http://www.net-.gov.bc.ca/NNR/NNR_UserGuide.html#1, Apr. 1997, pp. 1–17.

"Network Node Registry–Access Control Lists", http://www.net.gov.bc.ca/NN/NNR_AL_doc.html, Apr. 1997, pp. 1–5.

"Quality of Service Policy Propogation via Border Gateway Protocol", Cisco Systems, Inc., Feb. 1998, pp. 1–16.

"Distributed Weighted Random Early Detection", Cisco Systems, Inc., Feb. 1998, pp. 1–6.

"Distributed Weighted Fair Queuing", Cisco Systems, Inc., Mar. 1998, pp. 1–18.

Action Request Systems, Remedy Corporation, (c) 1998, pp. 1–6.

"3Com's Framework for Delivering Policy–Powered Networks", 3Com Corporation, Jun. 23, 1998.

Policy–Based Routing, Cisco Systems, Inc., Sep. 12, 1996, pp. 1–6.

Braden, Ed R., et al., "Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification", RFC 2205, Sep. 1997, pp. 1–47.

Internet Firewall Technologies, Firewall–1 Overview, Chapter 1, Jul. 1997, pp. 1–17.

FloodGate–1 Data TM Data Sheet, Enterprise Traffic Management, (co) 1997, pp. 1–7 and p. 1 of 1.

Isomaki, M., Differentiated Services for the Internet, May 9, 1998, pp. 1–17.

* cited by examiner

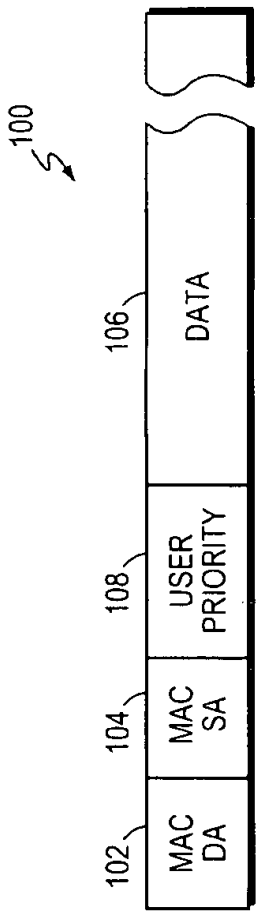


FIG. 1A (PRIOR ART)

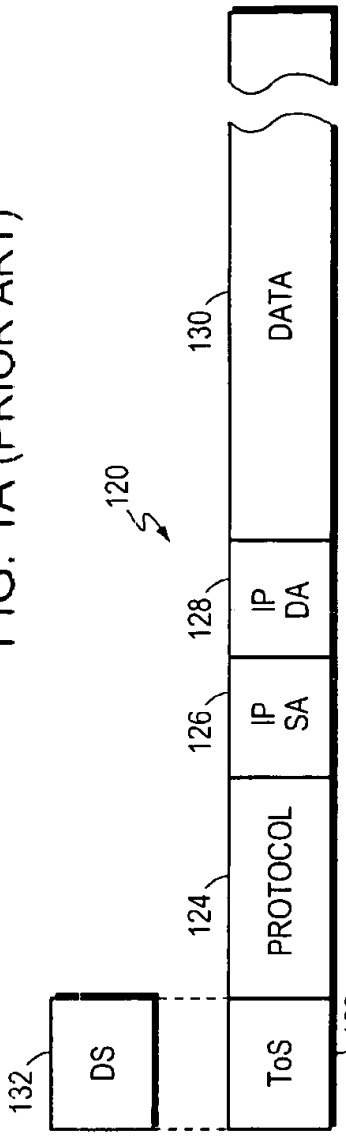


FIG. 1B (PRIOR ART)

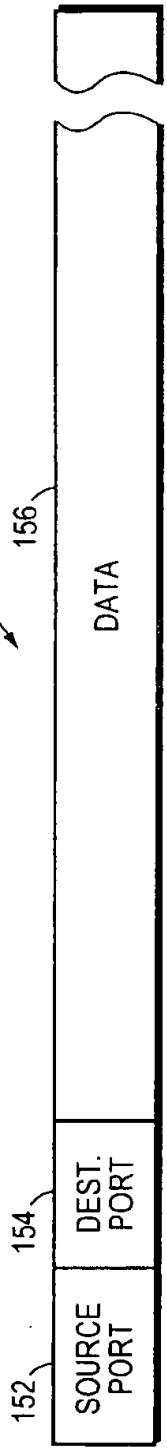


FIG. 1C (PRIOR ART)

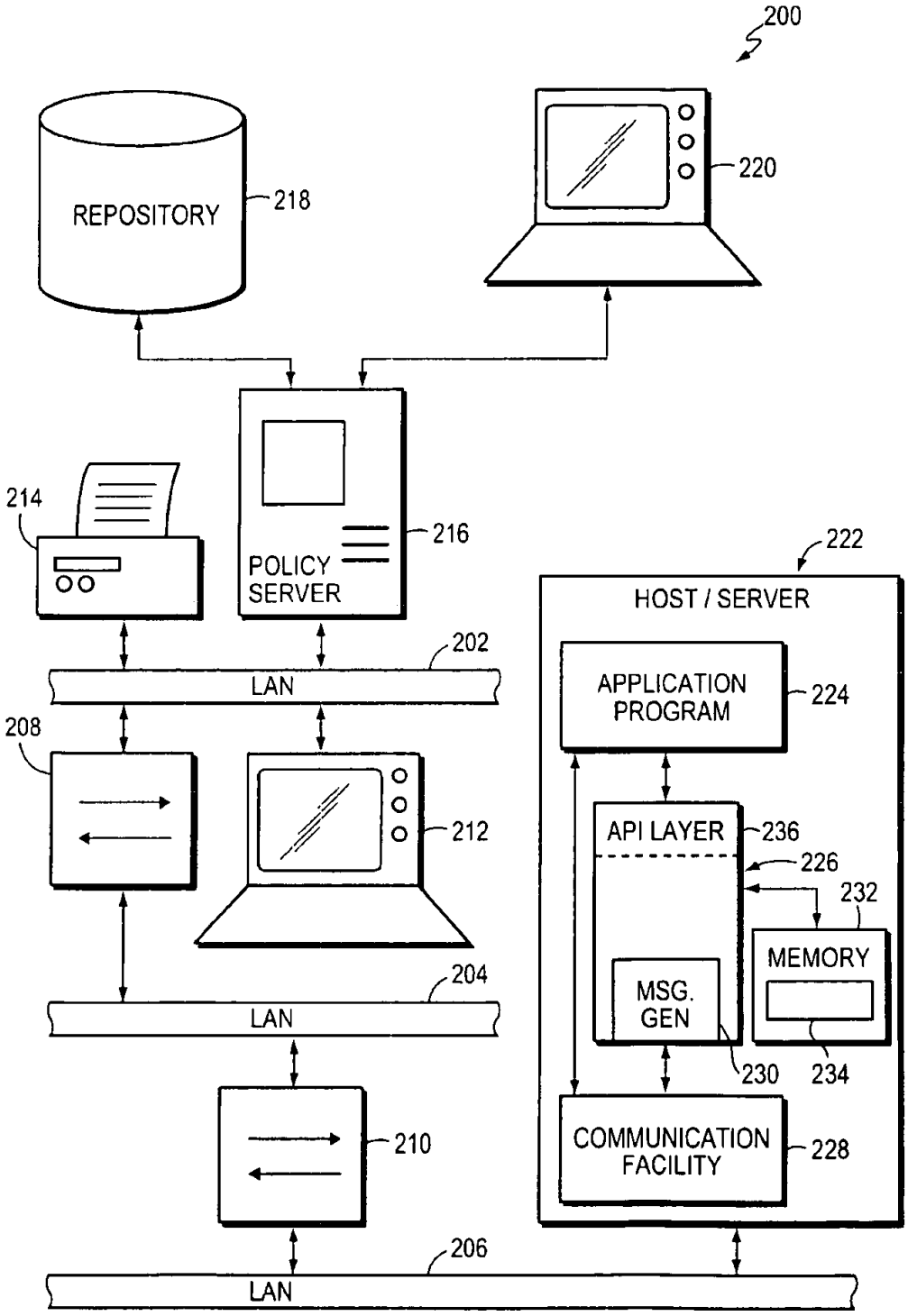

200
REPOSITORY
218
220
214
POLICY SERVER
216
222
HOST / SERVER
202
LAN
APPLICATION PROGRAM
224
208
212
API LAYER
236
226
232
MEMORY
MSG. GEN
230
234
204
LAN
210
COMMUNICATION FACILITY
228
206
LAN

FIG. 2

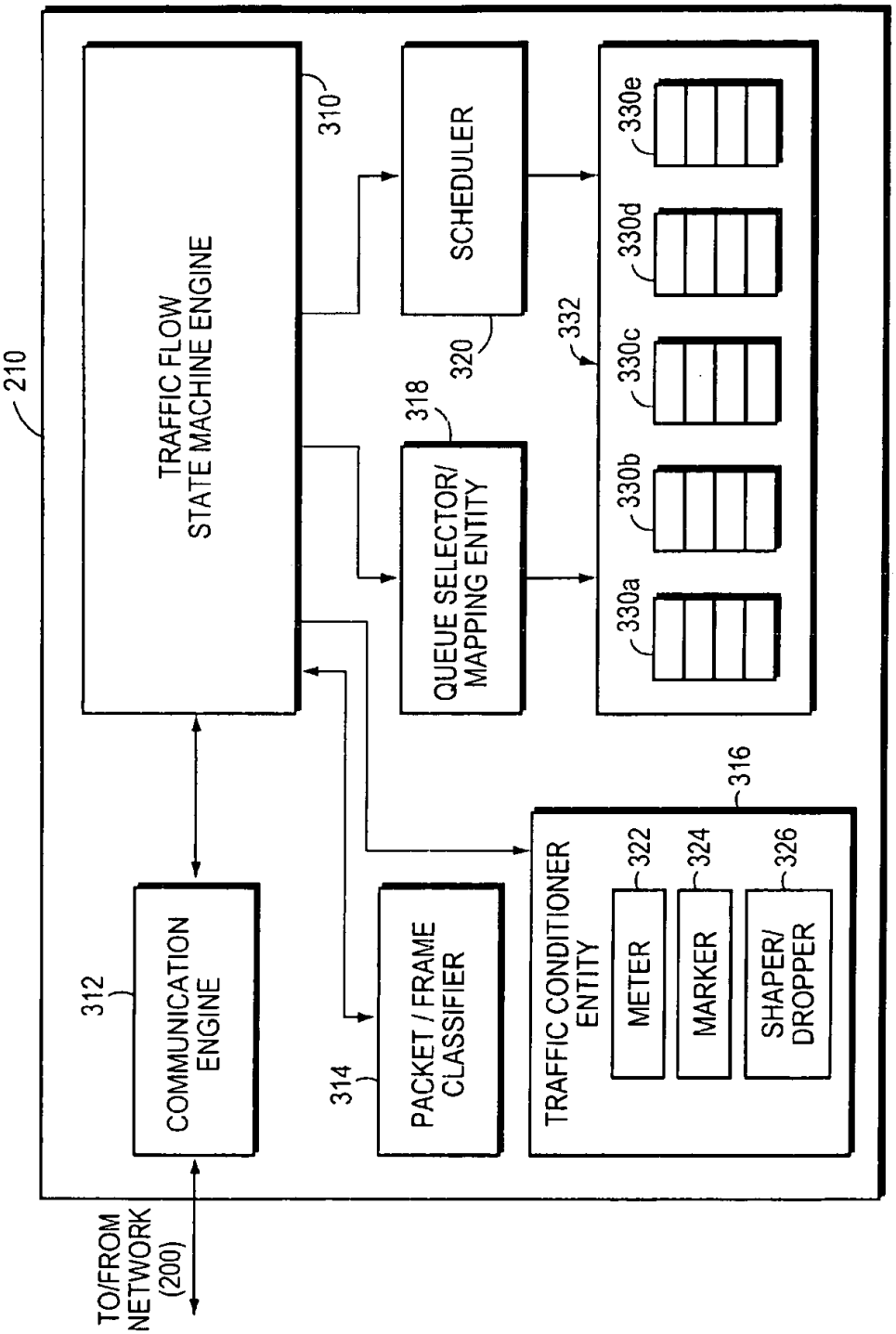

210
TRAFFIC FLOW STATE MACHINE ENGINE
310
SCHEDULER
320
332
330e
330d
330c
330b
330a
318
QUEUE SELECTOR/ MAPPING ENTITY
316
TRAFFIC CONDITIONER ENTITY
METER
322
MARKER
324
SHAPER/ DROPPER
326
312
COMMUNICATION ENGINE
314
PACKET / FRAME CLASSIFIER
TO/FROM NETWORK (200)

FIG. 3

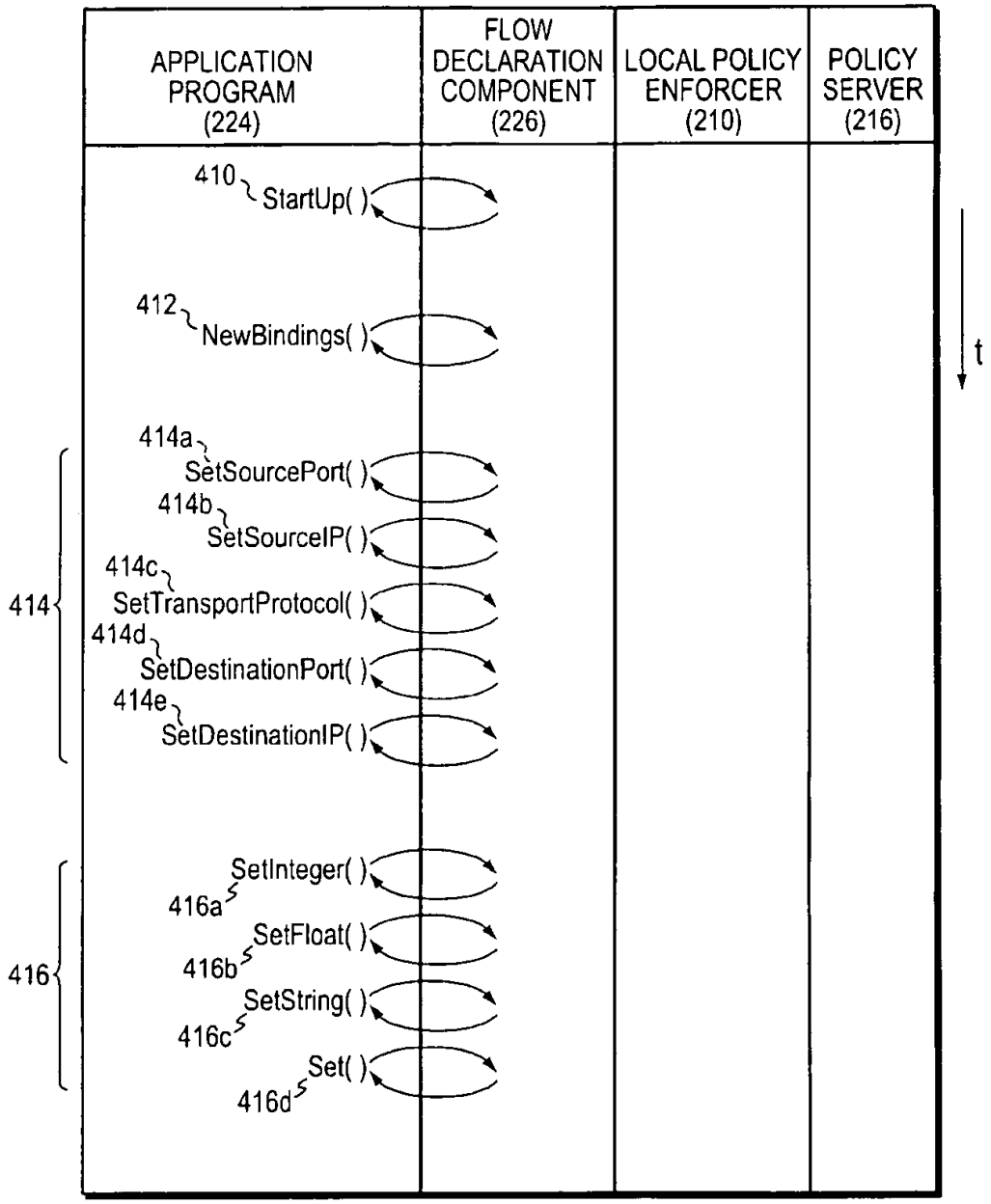

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
410
StartUp( )
412
NewBindings( )
414
414a
SetSourcePort( )
414b
SetSourceIP( )
414c
SetTransportProtocol( )
414d
SetDestinationPort( )
414e
SetDestinationIP( )
416
SetInteger( )
416a
SetFloat( )
416b
SetString( )
416c
Set( )
416d
t

FIG. 4A

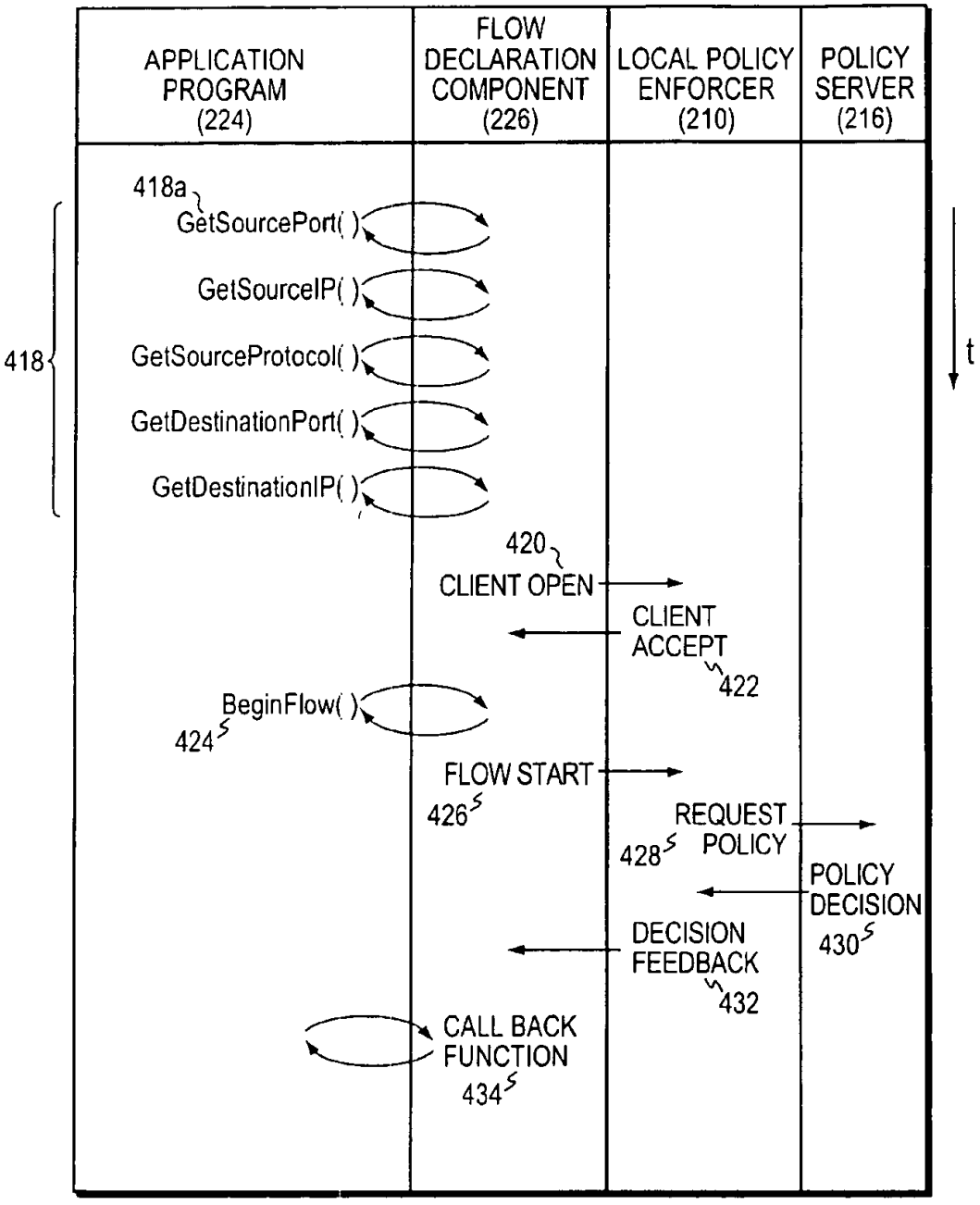

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
418a
GetSourcePort( )
GetSourceIP( )
GetSourceProtocol( )
GetDestinationPort( )
GetDestinationIP( )
418
t
420
CLIENT OPEN
CLIENT ACCEPT
422
BeginFlow( )
424
FLOW START
426
REQUEST POLICY
428
POLICY DECISION
430
DECISION FEEDBACK
432
CALL BACK FUNCTION
434

FIG. 4B

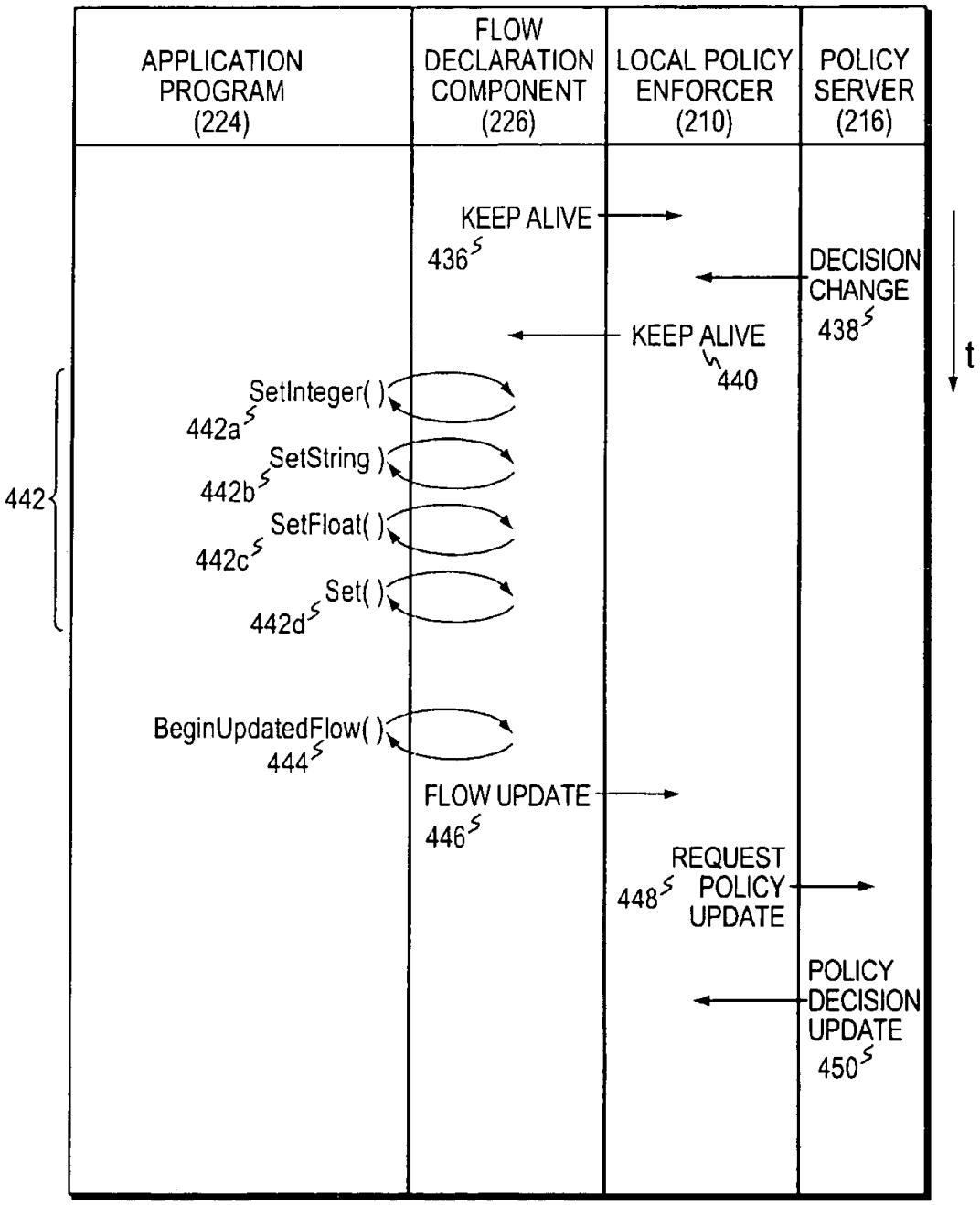

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
KEEP ALIVE
436
DECISION CHANGE
438
KEEP ALIVE
440
t
442
SetInteger( )
442a
SetString)
442b
SetFloat( )
442c
Set( )
442d
BeginUpdatedFlow( )
444
FLOW UPDATE
446
REQUEST POLICY UPDATE
448
POLICY DECISION UPDATE
450

FIG. 4C

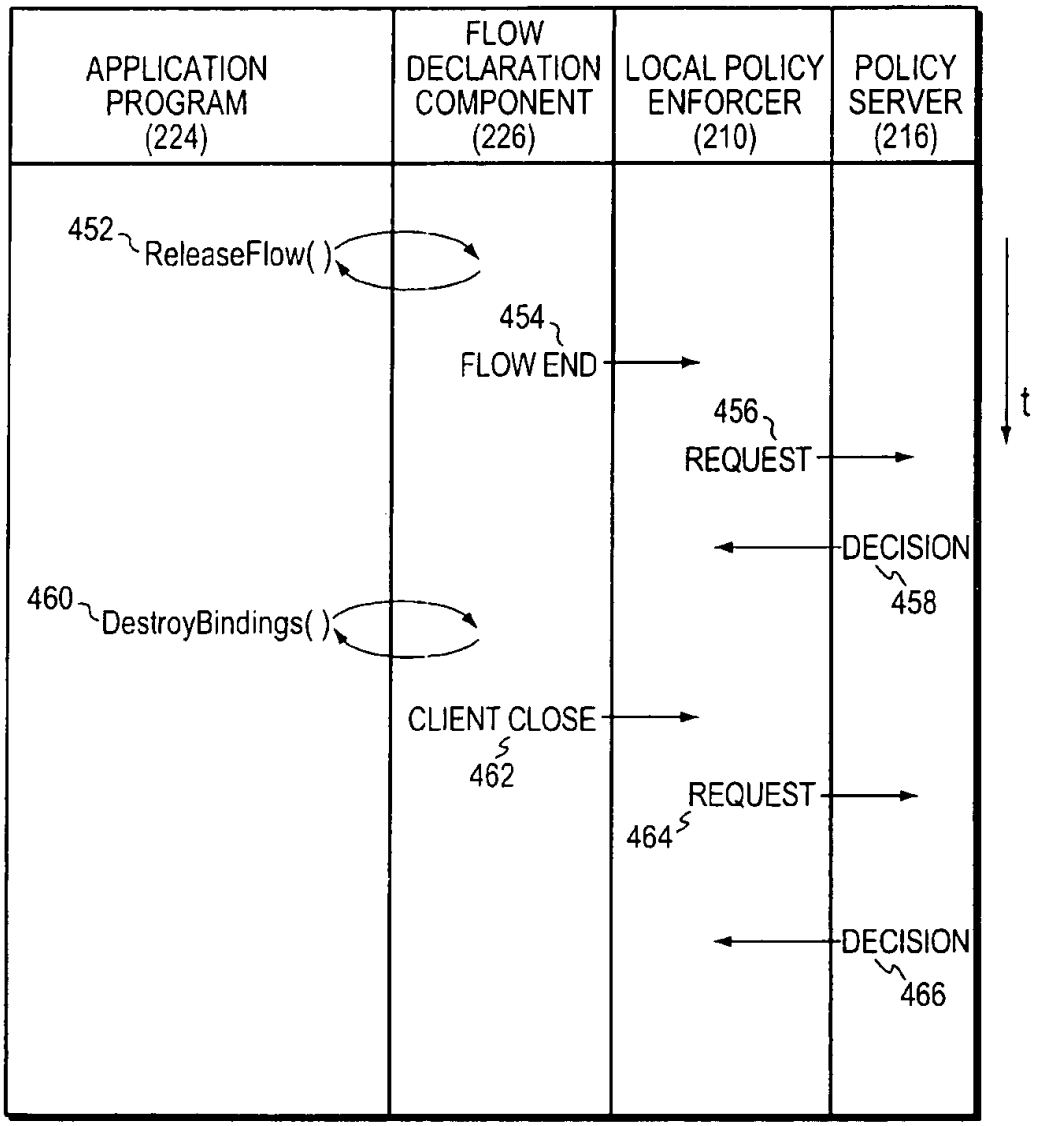

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
452
ReleaseFlow( )
454
FLOW END
456
REQUEST
DECISION
458
460
DestroyBindings( )
CLIENT CLOSE
462
REQUEST
464
DECISION
466
t

FIG. 4D

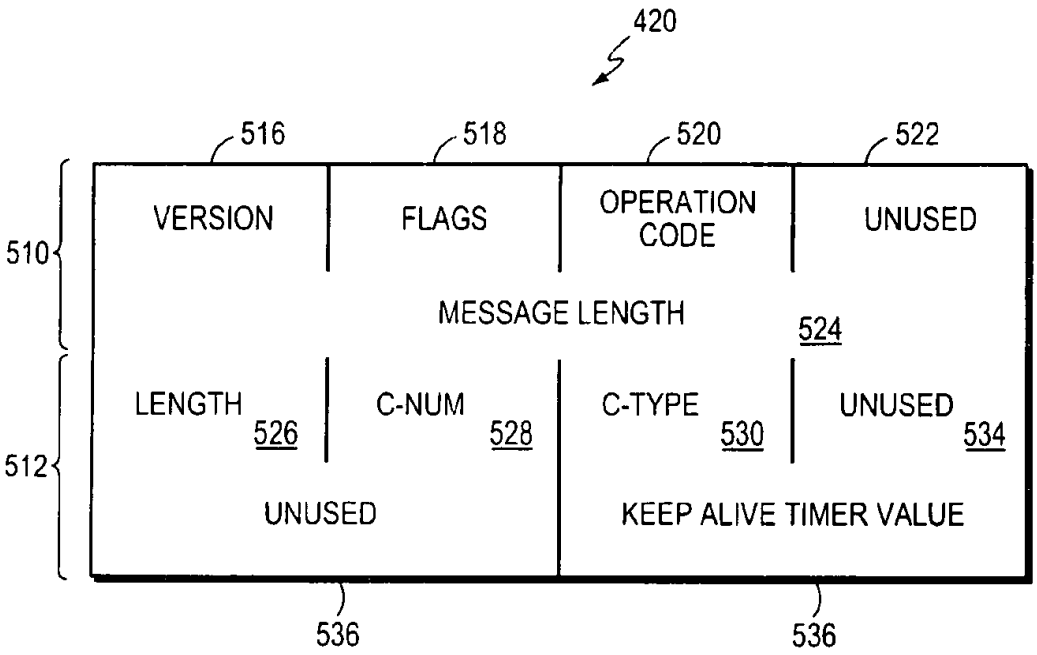

420
516
518
520
522
510
VERSION
FLAGS
OPERATION CODE
UNUSED
MESSAGE LENGTH
524
512
LENGTH
526
C-NUM
528
C-TYPE
530
UNUSED
534
UNUSED
KEEP ALIVE TIMER VALUE
536
536

FIG. 5A

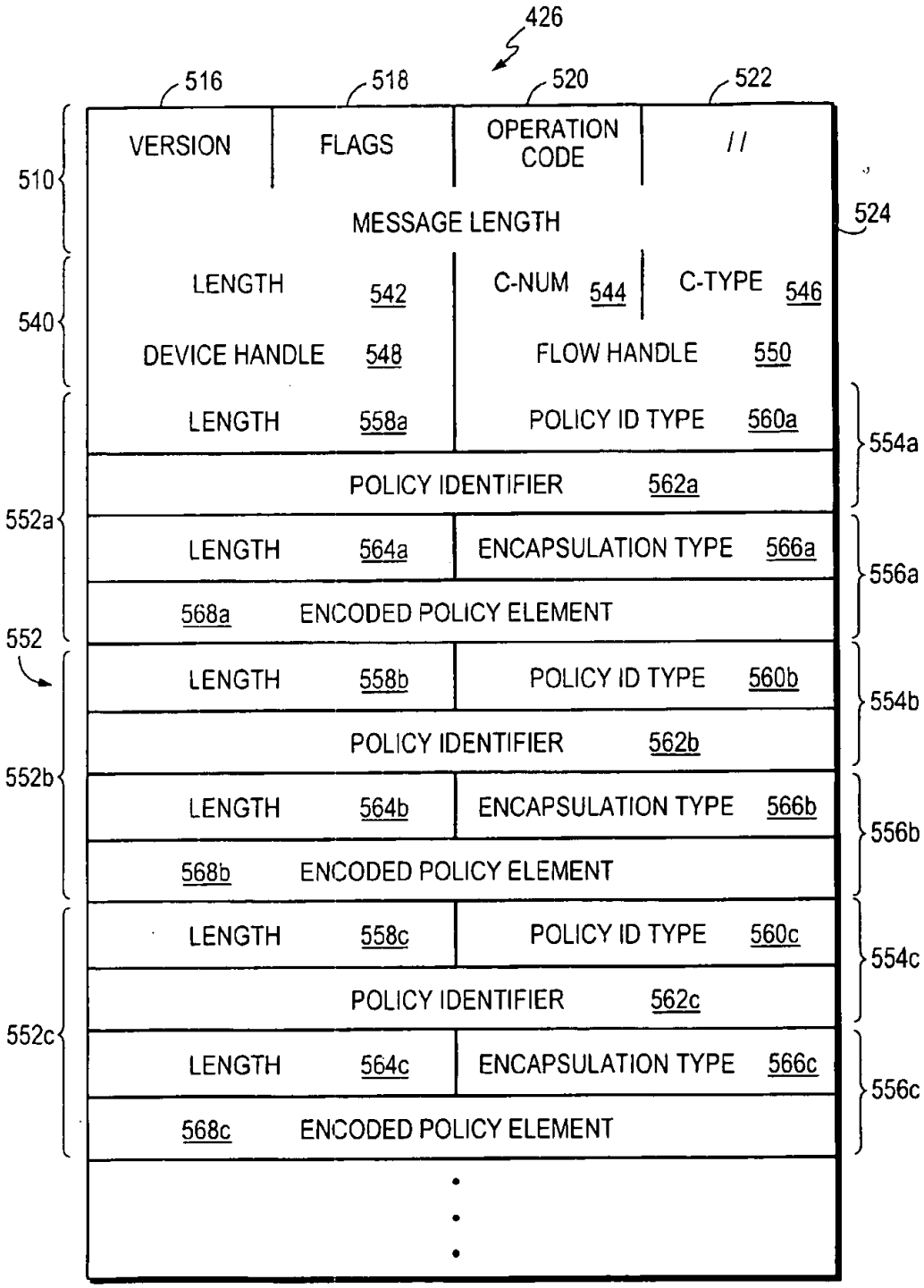

426
516
518
520
522
VERSION
FLAGS
OPERATION CODE
//
510
MESSAGE LENGTH
524
LENGTH 542
C-NUM 544
C-TYPE 546
540
DEVICE HANDLE 548
FLOW HANDLE 550
LENGTH 558a
POLICY ID TYPE 560a
554a
POLICY IDENTIFIER 562a
552a
LENGTH 564a
ENCAPSULATION TYPE 566a
556a
568a ENCODED POLICY ELEMENT
552
LENGTH 558b
POLICY ID TYPE 560b
554b
POLICY IDENTIFIER 562b
552b
LENGTH 564b
ENCAPSULATION TYPE 566b
556b
568b ENCODED POLICY ELEMENT
LENGTH 558c
POLICY ID TYPE 560c
554c
POLICY IDENTIFIER 562c
552c
LENGTH 564c
ENCAPSULATION TYPE 566c
556c
568c ENCODED POLICY ELEMENT

FIG. 5B

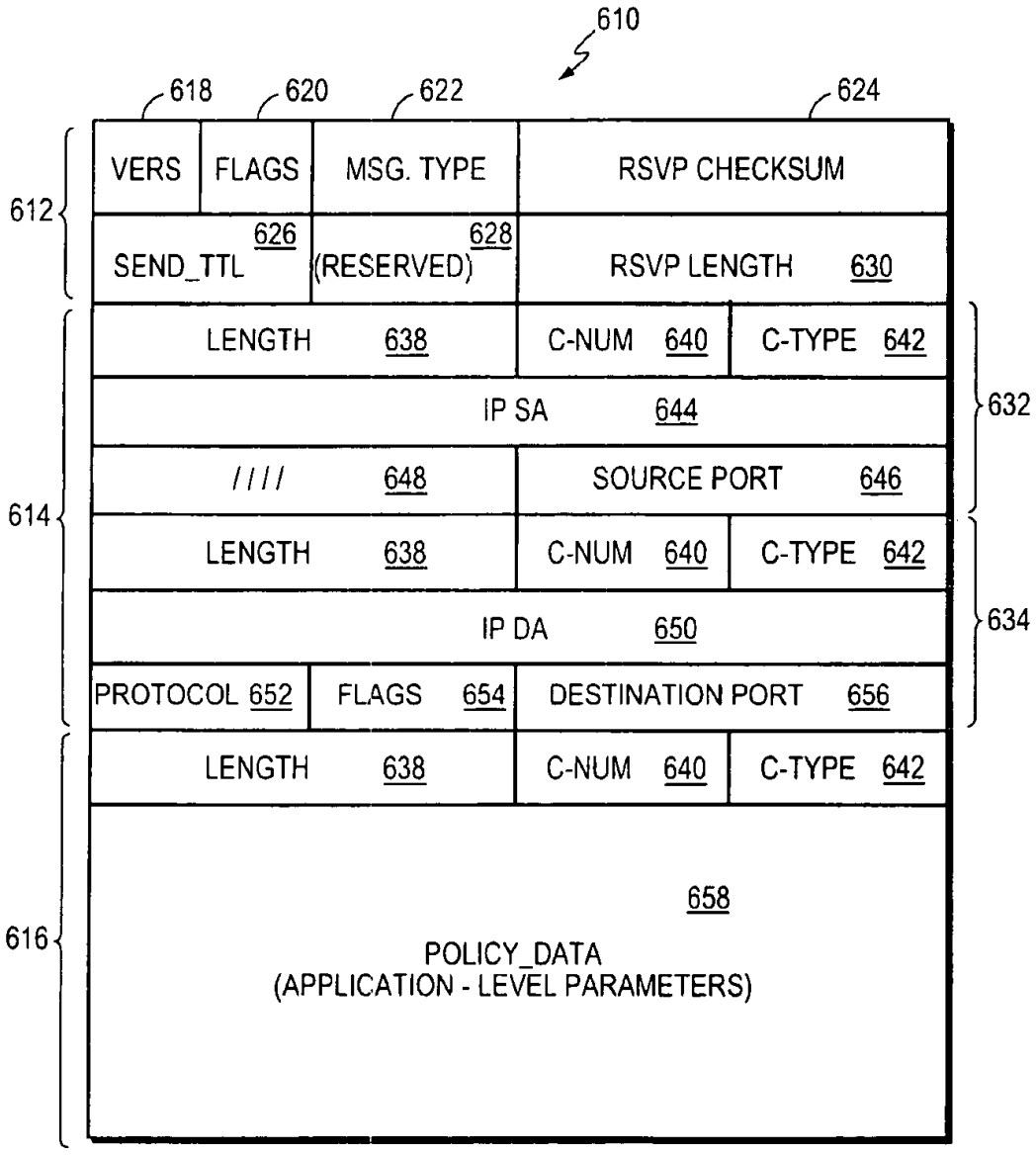

610
618
620
622
624
VERS
FLAGS
MSG. TYPE
RSVP CHECKSUM
612
SEND_TTL 626
(RESERVED) 628
RSVP LENGTH 630
LENGTH 638
C-NUM 640
C-TYPE 642
IP SA 644
632
//// 648
SOURCE PORT 646
614
LENGTH 638
C-NUM 640
C-TYPE 642
IP DA 650
634
PROTOCOL 652
FLAGS 654
DESTINATION PORT 656
LENGTH 638
C-NUM 640
C-TYPE 642
658
POLICY_DATA
(APPLICATION - LEVEL PARAMETERS)
616

FIG. 6

METHOD AND APPARATUS FOR IDENTIFYING NETWORK DATA TRAFFIC FLOWS AND FOR APPLYING QUALITY OF SERVICE TREATMENTS TO THE FLOWS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of application Ser. No. 09/911,122, filed Jul. 23, 2001, now U.S. Pat. No. 6,434,624, which is a continuation of application Ser. No. 09/206,067, filed Dec. 4, 1998, now U.S. Pat. No. 6,286,052, which is hereby incorporated by reference in its entirety.

This application is related to the following U.S. patent application:

U.S. patent application Ser. No. 09/179,036 entitled, METHOD AND APPARATUS FOR DEFINING AND IMPLEMENTING HIGH-LEVEL QUALITY OF SERVICE POLICIES IN COMPUTER NETWORKS, filed Oct. 26,1998, now U.S. Pat. No. 6,167,445 and assigned to the assignee of the present application.

FIELD OF THE INVENTION

The present invention relates generally to computer networks, and more specifically, to a method and apparatus for identifying network data traffic flows and for to applying quality of service or policy treatments thereto.

BACKGROUND OF THE INVENTION

A computer network typically comprises a plurality of interconnected entities that transmit (i.e., "source") or receive (i.e., "sink") data frames. A common type of computer network is a local area network ("LAN") which typically refers to a privately owned network within a single building or campus. LANs employ a data communication protocol (LAN standard), such as Ethernet, FDDI or Token Ring, that defines the functions performed by the data link and physical layers of a communications architecture (i.e., a protocol stack), such as the Open Systems Interconnection (OSI) Reference Model. In many instances, multiple LANs may be interconnected by point-to-point links, microwave transceivers, satellite hook-ups, etc. to form a wide area network ("WAN"), metropolitan area network ("MAN") or intranet. These LANs and/or WANs, moreover, may be coupled through one or more gateways to the Internet.

Each network entity preferably includes network communication software, which may operate in accordance with the well-known Transport Control Protocol/Internet Protocol (TCP/IP). TCP/IP basically consists of a set of rules defining how entities interact with each other. In particular, TCP/IP defines a series of communication layers, including a transport layer and a network layer. At the transport layer, TCP/IP includes both the User Data Protocol (UDP), which is a connectionless transport protocol, and TCP which is a reliable, connection-oriented transport protocol. When a process at one network entity wishes to communicate with another entity, it formulates one or more messages and passes them to the upper layer of the TCP/IP communication stack. These messages are passed down through each layer of the stack where they are encapsulated into packets and frames. Each layer also adds information in the form of a header to the messages. The frames are then transmitted over the network links as bits. At the destination entity, the bits are re-assembled and passed up the layers of the destination entity's communication stack. At each layer, the corresponding message headers are also stripped off, thereby recovering the original message which is handed to the receiving process.

One or more intermediate network devices are often used to couple LANs together and allow the corresponding entities to exchange information. For example, a bridge may be used to provide a "bridging" function between two or more LANs. Alternatively, a switch may be utilized to provide a "switching" function for transferring information, such as data frames or packets, among entities of a computer network. Typically, the switch is a computer having a plurality of ports that couple the switch to several LANs and to other switches. The switching function includes receiving data frames at a source port and transferring them to at least one destination port for receipt by another entity. Switches may operate at various levels of the communication stack. For example, a switch may operate at layer **2** which, in the OSI Reference Model, is called the data link layer and includes the Logical Link Control (LLC) and Media Access Control (MAC) sub-layers.

Other intermediate devices, commonly referred to as routers, may operate at higher communication layers, such as layer **3**, which in TCP/IP networks corresponds to the Internet Protocol (IP) layer. IP data packets include a corresponding header which contains an IP source address and an IP destination address. Routers or layer **3** switches may re-assemble or convert received data frames from one LAN standard (e.g., Ethernet) to another (e.g. Token Ring). Thus, layer **3** devices are often used to interconnect dissimilar subnetworks. Some layer **3** intermediate network devices may also examine the transport layer headers of received messages to identify the corresponding TCP or UDP port numbers being utilized by the corresponding network entities. Many applications are assigned specific, fixed TCP and/or UDP port numbers in accordance with Request for Comments (RFC) **1700**. For example, TCP/UDP port number **80** corresponds to the hyper text transport protocol (HTTP), while port number **21** corresponds to file transfer protocol (ftp) service.

Allocation of Network Resources

Computer networks include numerous services and resources for use in moving traffic throughout the network. For example, different network links, such as Fast Ethernet, Asynchronous Transfer Mode (ATM) channels, network tunnels, satellite links, etc., offer unique speed and bandwidth capabilities. Particular intermediate devices also include specific resources or services, such as number of priority queues, filter settings, availability of different queue selection strategies, congestion control algorithms, etc.

Individual frames or packets, moreover, can be marked so that intermediate devices may treat them in a predetermined manner. For example, the Institute of Electrical and Electronics Engineers (IEEE), in an appendix (802.1p) to the 802.1D bridge standard, describes additional information for the MAC header of Data Link Layer frames. FIG. **1A** is a partial block diagram of a Data Link frame **100** which includes a MAC destination address (DA) field **102**, a MAC source address (SA) field **104** and a data field **106**. In accordance with the 802.1Q standard, a user_priority field **108**, among others, is inserted after the MAC SA field **104**. The user_priority field **108** may be loaded with a predetermined value (e.g., 0–7) that is associated with a particular treatment, such as background, best effort, excellent effort, etc. Network devices, upon examining the user_priority field **108** of received Data Link frames **100**, apply the corresponding treatment to the frames. For example, an intermediate device may have a plurality of transmission priority queues per port, and may assign frames to different queues of a destination port on the basis of the frame's user priority value.

FIG. 1B is a partial block diagram of a Network Layer packet **120** corresponding to the Internet Protocol. Packet **120** includes a type_of_service (ToS) field **122**, a protocol field **124**, an IP source address (SA) field **126**, an IP destination address (DA) field **128** and a data field **130**. The ToS field **122** is used to specify a particular service to be applied to the packet **120**, such as high reliability, fast delivery, accurate delivery, etc., and comprises a number of sub-fields (not shown). The sub-fields include a three bit IP precedence (IPP) field and three one bit flags (Delay, Throughput and Reliability). By setting the various flags, an entity may indicate which overall service it cares most about (e.g., Throughput versus Reliability). Version 6 of the Internet Protocol (IPv6) similarly defines a traffic class field, which is also intended to be used for defining the type of service to be applied to the corresponding packet.

Recently, a working group of the Internet Engineering Task Force (IETF), which is an independent standards organization, has proposed replacing the ToS field **112** of Network Layer packets **120** with a one octet differentiated services (DS) field **132** that can be loaded with a differentiated services codepoint. Layer **3** devices that are DS compliant apply a particular per-hop forwarding behavior to data packets based on the contents of their DS fields **132**. Examples of per-hop forwarding behaviors include expedited forwarding and assured forwarding. The DS field **132** is typically loaded by DS compliant intermediate devices located at the border of a DS domain, which is a set of DS compliant intermediate devices under common network administration. Thereafter, interior DS compliant devices along the path simply apply the corresponding forwarding behavior to the packet **120**.

FIG. 1C is a partial block diagram of a Transport Layer packet **150**. The network layer packet **150** preferably includes a source port field **152**, a destination port field **154** and a data field **156**, among others. Fields **152** and **154** are preferably loaded with the predefined or dynamically agreed-upon TCP or UDP port numbers being utilized by the corresponding network entities.

Service Level Agreements

To interconnect dispersed computer networks, many organizations rely on the infrastructure and facilities of internet service providers (ISPs). For example, an organization may lease a number of T1 lines to interconnect various LANs. These organizations and ISPs typically enter into service level agreements, which include one or more traffic specifiers. These traffic specifiers may place limits on the amount of resources that the subscribing organization will consume for a given charge. For example, a user may agree not to send traffic that exceeds a certain bandwidth (e.g., 1 Mb/s). Traffic entering the service provider's network is monitored (i.e., "policed") to ensure that it complies with the relevant traffic specifiers and is thus "in-profile". Traffic that exceeds a traffic specifier (i.e., traffic that is "out-of-profile") may be dropped or shaped or may cause an accounting change (i.e., causing the user to be charged a higher rate). Another option is to mark the traffic as exceeding the traffic specifier, but nonetheless allow it to proceed through the network. If there is congestion, an intermediate network device may drop such "marked" traffic first in an effort to relieve the congestion.

Filtered

CLM 3

Multiple Traffic Flows

A process executing at a given network entity, moreover, may generate hundreds if not thousands of traffic flows that are transmitted across the corresponding network every day. A traffic flow generally refers to a set of messages (frames and/or packets) that typically correspond to a particular task, transaction or operation (e.g., a print transaction) and may be identified by **5** network and transport layer parameters (e.g., source and destination IP addresses, source and destination TCP/UDP port numbers and transport protocol). Furthermore, the treatment that should be applied to these different traffic flows varies depending on the particular traffic flow at issue. For example, an online trading application may generate stock quote messages, stock transaction messages, transaction status messages, corporate financial information messages, print messages, data back-up messages, etc. A network administrator, moreover, may wish to have very different policies or service treatments applied to these various traffic flows. In particular, the network administrator may want a stock quote message to be given higher priority than a print transaction. Similarly, a $1 million stock transaction message for a premium client should be assigned higher priority than a $100 stock transaction message for a standard customer. Most intermediate network devices, however, lack the ability to distinguish among multiple traffic flows, especially those originating from the same host or server.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method and apparatus for identifying one or more traffic flows from a source entity.

It is a further object of the present invention to provide a method and apparatus for obtaining traffic policies to be applied to identified traffic flows.

It is a further object of the present invention to manage traffic flows in accordance with corresponding policies.

Briefly, the invention relates to a method and apparatus for identifying specific traffic flows originating from a network entity and for applying predetermined policy or service treatments to those flows. In particular, a network entity includes a flow declaration component that is coupled to one or more application programs executing on the entity. The network entity also includes a communication facility that supports to message exchange between the application program and other network entities. The flow declaration component includes a message generator and an associated memory for storing one or more traffic flow data structures. For a given traffic flow, the application program calls the flow declaration component and provides it with one or more identifying parameters corresponding to the given flow. In particular, the application program may provide network and transport layer parameters, such as IP source and destination addresses, TCP/UDP port numbers and transport protocol associated with the given traffic flow. It also provides one or more application-level parameters, such as a transaction-type (e.g., a stock transaction), a sub-transaction-type (e.g., a $1 Million stock purchase order), etc. The flow declaration component provides this information to a local policy enforcer, which, in turn, may query a policy server to obtain one or more policy or service treatments that are to be applied to the identified traffic flow. The local policy enforcer then monitors the traffic originating from the network entity and, by examining IP source and destination addresses, among other information, applies the prescribed policy or service treatments to the given traffic flow.

In the preferred embodiment, the application program and the flow declaration component at the network entity interact

through an Application Programming Interface (API) layer, which includes a plurality of system calls. In addition, the flow declaration component generates and transmits one or more application parameter declaration (APD) messages to the local policy enforcer. The APD messages contain the network and transport layer parameters (e.g., IP source and destination addresses, TCP/UDP port numbers and transport protocol) stored at the traffic flow data structure for the given flow. The messages may also contain the application-level parameters specified by the application program. The information, moreover, may be in the form of objects generated by the flow declaration component. Preferably, the flow declaration component and the local policy enforcer exchange messages in accordance with a novel protocol that defines a message scheme in addition to a message format. The local policy enforcer and the policy server may utilize the Common Open Policy Service (COPS) protocol to request and receive particular policies or service treatment rules. Preferably, the policy server maintains or otherwise has access to a store of network policies established by the network administrator.

In another aspect of the invention, the local policy enforcer may establish a traffic flow state that includes the policy or service treatments specified by the policy server. It then monitors the traffic flows originating from the network entity looking for the given traffic flow. Once the given traffic flow is identified, the local policy enforcer applies the policy or service treatments set forth in the corresponding traffic flow state. For example, the policy enforcer may mark the packets or frames with a high priority DS codepoint. When the given traffic flow is complete, the application program may notify the flow declaration component, which, in turn, signals the end of the traffic flow to the local policy enforcer. The policy enforcer may request authorization from the policy server to release or otherwise discard the respective traffic flow state.

In an alternative embodiment of the invention, policy rules may be cached at the local policy enforcer to eliminate the need to query the policy server for each new traffic flow.

In another embodiment of the invention, the APD messages are replaced with one to or more enhanced Path or Reservation messages as originally specified in the Resource ReSerVation Protocol (RSVP).

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings, in which:

FIGS. 1A–1C, previously discussed, are partial block diagram of network messages;

FIG. **2** is a highly schematic block diagram of a computer network;

FIG. **3** is a highly schematic, partial block diagram of local policy enforcer;

FIGS. **4A–4D** are flow diagrams illustrating the message scheme and tasks performed in identifying a traffic flow and obtaining the corresponding policies;

FIGS. **5A–5B** are highly schematic block diagrams illustrating the preferred format of an application parameter declaration message; and

FIG. **6** is a highly schematic block diagram illustrating an enhanced Resource ReSerVation Protocol (RSVP) message in accordance with the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. **2** is a highly schematic block diagram of a computer network **200**. The network **200** includes a plurality of local area networks (LANS) **202**, **204** and **206** that are interconnected by a plurality of intermediate network devices **208**, **210**. Coupled to the LANs are a plurality of entities, such as end station **212** and print server **214**. The network further includes at least one policy server **216** that may be coupled to a repository **218** and to a network administrator's station **220**. A server suitable for use as policy server **216** is any Intel x86/Windows NT® or Unix-based platform. The network **200** also includes at least one host or server **222** configured in accordance with the present invention.

In particular, the host/server **222** includes at least one application program or process **224**, a flow declaration component **226** and a communication facility **228**. The is flow declaration component **226** includes a message generator **230** that is in communicating relation with the communication facility **228**. Component **226** is also coupled to an associated memory **232** for storing one or more traffic flow data structures **234**. The application program **224** is in communicating relation with both the communication facility **228** and, through an Application Programming Interface (API) layer **236**, to the flow declaration component **226**. The communication facility **228**, in turn, is connected to network **200** via LAN **206**. The host/server **222** also comprises conventional programmable processing elements (not shown), which may contain software program instructions pertaining to the methods of the present invention. Other computer readable media may also be used to store the program instructions.

The communication facility **228** preferably includes one or more software libraries for implementing a communication protocol stack allowing host/server **222** to exchange messages with other network entities, such as end station **212**, print server **214**, etc. In particular, the communication facility **228** may include software layers corresponding to the Transmission Control Protocol/Internet Protocol (TCP/IP), the Internet Packet Exchange (IPX) protocol, the AppleTalk protocol, the DECNet protocol and/or NetBIOS Extended User Interface (NetBEUI). Communication facility **228** further includes transmitting and receiving circuitry and components, including one or more network interface cards (NICs) that establish one or more physical ports to LAN **206** or other LANs for exchanging data packets and frames.

Intermediate network devices **208**, **210** provide basic bridging functions including filtering of data traffic by medium access control (MAC) address, "learning" of a MAC address based upon a source MAC address of a frame and forwarding of the frame based upon a destination MAC address or route information field (RIF). They may also include an Internet Protocol (IP) software layer and provide route processing, path determination and path switching functions. In the illustrated embodiment, the intermediate network devices **208**, **210** are computers having transmitting and receiving circuitry and components, including network interface cards (NICs) establishing physical ports, for exchanging data frames. Intermediate network device **210**, moreover, is preferably configured as a local policy enforcer for traffic flows originating from host/server **222**, as described below.

Filtering

It should be understood that the network configuration **200** of FIG. **2** is for illustrative purposes only and that the present invention will operate with other, possibly far more complex, network topologies. For example, the repository **218** and network administrator's station **220** may be directly or indirectly connected to the policy server **216** (e.g., through one or more intermediate devices).

FIG. **3** is a partial block diagram of local policy enforcer **210**. Local policy enforcer **210** includes a traffic flow state

machine engine **310** for maintaining flow states corresponding to host/server **222** traffic flows, as described below. The traffic flow state machine engine **310** is coupled to a communication engine **312**. The communication engine **312** is configured to formulate and exchange messages with the policy server **216** and the flow declaration component **226** at host/server **222**. That is, communication engine **312** includes or has access to conventional circuitry for transmitting and receiving messages over the network **200**. The traffic flow state machine engine **310** is also coupled to several traffic management resources and mechanisms. In particular, traffic flow state machine engine **310** is coupled to a packet/frame classifier **314**, a traffic conditioner entity **316**, a queue selector/mapping entity **318** and a scheduler **320**. The traffic conditioner entity **316** includes several sub-components, including one or more metering entities **322**, one or more marker entities **324**, and one or more shaper/dropper entities **326**. The queue selector/mapping entity **318** and scheduler **320** operate on the various queues established by local policy enforcer **210** for its ports and/or interfaces, such as queues **330***a*–**330***e* corresponding to an interface **332**.

The term intermediate network device is intended broadly to cover any intermediate device for interconnecting end stations of a computer network, including, without limitation, layer 3 devices or routers, as defined by Request for Comments (RFC) **1812** from the Internet Engineering Task Force (IETF), intermediate devices that are only partially compliant with RFC **1812**, intermediate devices that provide additional functionality, such as Virtual Local Area Network (VLAN) support, IEEE 802.1Q support and/or IEEE 802.1D support, etc. Intermediate network device also includes layer **2** intermediate devices, such as switches and bridges, including, without limitation, devices that are fully or partially compliant with the IEEE 802.1D standard and intermediate devices that provide additional functionality, such as VLAN support, IEEE 802.1Q support and/or IEEE 802.1p support, Asynchronous Transfer Mode (ATM) switches, Frame Relay switches, etc.

FIGS. **4**A–**4**D are flow diagrams illustrating a preferred message scheme, relative to time t, in accordance with the present invention. In general, application program **224** identifies one or more anticipated traffic flows to the flow declaration component **226**, which, in turn, notifies the local policy enforcer **210**. The local policy enforcer **210** requests and receives from the policy server **216** corresponding policy or service treatments for the anticipated traffic flows. Local policy enforcer **210** then monitors the traffic originating from host/server **222** to identify those frames and/or packets corresponding to the identified flows. When such a flow is detected, local policy enforcer **210** applies the specified policy or service treatments to corresponding data frames and/or packets.

Identification of Traffic Flows

Assume that application program **224** is a stock transaction program that can provide stock quotes to and process stock transactions from remote clients, such as end station **212**. The application program **224** preferably communicates with end station **212** across network **200** through the communication facility **228** at host/server **222** in a conventional manner. Program **224** also communicates with the flow declaration component **226** preferably through a plurality of application programming interface (API) system calls to API layer **236**. These API calls are generally issued by the program **224** along with one or more arguments and may be returned by the flow declaration component **226**.

In particular, upon initialization at host/server **222**, the application program **224** preferably issues a StartUp( ) API call **410** to the API layer **236** at flow declaration component **226**. Program **226** preferably loads the StartUp( ) call **410** with an application identifier that uniquely identifies application program **224** to component **226** as an argument. The application identifier may be a globally unique identifier (GUID), which is a **128** bit long value typically provided by the application developer, although other identifiers may also be used (e.g., application name). The StartUp( ) call **410** may be returned by the flow declaration component **226** with a version number as an argument. The version number corresponds to the version of software being executed by the flow declaration component **226**. Other arguments, such as the quality-of-service (QoS) and/or traffic management resources that are available to traffic flows originating from program **224**, may also be returned by flow declaration component **226**.

For example, assume end station **212** contacts program **224** and requests a stock quote for a particular equity (e.g., IBM common stock). Program **224** retrieves the requested information and prepares a message containing the requested stock quote for transmission to end station **212**. Before program **224** commences the traffic flow corresponding to requested stock quote, it preferably issues a NewBindings( ) call **412** to the API layer **236** of the flow declaration component **226**. The NewBindings( ) call **412** is used to inform flow declaration component **226** of an anticipated traffic flow to which some policy or service treatments should be applied. In response to the NewBindings( ) call **412**, flow declaration component **226** generates a bindings handle, e.g., H1, and creates a traffic flow data structure **234** within associated memory **232**. Component **226** also maps or associates the traffic flow data structure **234** with the returned bindings handle H1. Flow declaration component **226** also returns the NewBindings( ) call **412** to program **224** with the handle H1 as an argument.

Next, traffic flow data structure **234** is loaded with information identifying the anticipated traffic flow. More specifically, program **224** next issues one or more network and transport layer parameter "Set" API calls **414**. These Set calls **414** are used by the flow declaration component **226** to load traffic flow data structure **234** with network and transport layer parameters, such as Internet Protocol (IP) addresses and TCP/UDP port numbers. For example, program **224** may issue a SetSourcePort( ) call **414***a* using the returned handle, H1, and the transport layer port number (e.g., TCP port number **1098**) to be utilized by program **226** as its arguments. In response, flow declaration component **226** loads the identified source port number (i.e., **1098**) into the traffic flow data structure **234** corresponding to handle H1. Flow declaration component **226** may return an acknowledgment to program **224** as an argument to the SetSourcePort( ) call **414***a*. If a problem arises, flow declaration component **226** may return an error message (e.g., insufficient memory, unknown handle, out of bound port number, etc.) as the argument.

In a similar manner, program **224** preferably causes the flow declaration component **226** to load the corresponding traffic flow data structure **234** with its IP address, the transport layer protocol (e.g., TCP) and the destination port number and IP address of the receiving process at end station **212**. More specifically, in addition to the SetSourcePort( ) call **414***a*, program **224** may issue one or more of the following API system calls:

SetSourceIP( ) **414***b*;

SetTransportProtocol( ) **414***c*;

SetDestinationPort( ) **414***d*; and

SetDestinationIP( ) **414***e*.

Again, program **224** uses the previously returned handle, H1, and the corresponding information (e.g., IP address, transport protocol or port number) as arguments to these API

calls. As each Set API call **414** is received, the flow declaration component **226** loads the identified parameter into the traffic flow data structure **234**. Flow declaration component **226** may similarly return the Set API call **414** with an error code or an acknowledgment as an argument. It should be understood that additional "Set" API calls **414** may be defined depending on the format of the included information. For example, by utilizing a SetSourceIPByLong( ) call (not shown), program **224** may specify its IP address as a **32** bit binary sequence. Alternatively, by utilizing a SetSourceIPByString( ) call (not shown), program **224** may specify its IP address in dotted decimal format (e.g., **128.120.52.123**) or as a host name (e.g., name.department.company.domain). In addition, a single SetNetworkTransportParameters( ) system call may be defined to set all of the network and transport layer parameters at once.

It should be understood that application program **224** may obtain IP source and destination addresses, port numbers and transport protocol for use in communicating with end station **212** from the communication facility **228** in a conventional manner. It should be further understood that application program **224** may utilize one or more wildcards when specifying the network and transport layer parameters.

In addition to the network and transport layer parameters (e.g., source and destination IP addresses, transport protocol and source and destination TCP/UDP port numbers) which correspond to a particular flow of traffic, program **236** may specify other identifying characteristics and/or policy elements of the anticipated traffic flow. That is, program **224** may issue one or more application-level "Set" API calls **416** to the flow declaration component **226**. For example, a SetInteger( ) call **416***a* may be used to specify some numerical aspect (e.g., the size of a file being transferred) of the anticipated traffic flow. The arguments of the SetInteger( ) call **416***a* include the handle **H1**, the numeric policy element (e.g., 786 Kbytes) and a policy element identifier (PID) that maps the numeric policy element to a particular type or class of information (e.g., file size). When the traffic type data structure **234** is subsequently transferred to and processed by other entities, as described below, the PID will identify its corresponding information. In response to the SetInteger( ) call **416***a*, flow declaration component **226** loads the traffic flow data structure **234** with the numeric policy element and the PID. Flow declaration component **226** may return the SetInteger( ) call **416***a* to program **224** with an acknowledgment or error message as arguments.

Other application-level Set calls may also be defined. For example, a SetFloat( ) call **416***b* is used to associate a numeric value represented in floating decimal format with the anticipated traffic flow. A SetString( ) call **416***c* may be used to associate an alpha-numeric string with the anticipated flow. For example, if the anticipated traffic flow is to contain a video segment, program **224** may identify the name of the particular video segment and/or the viewer by utilizing the SetString( ) call **416***c*. Program **224** uses the handle **H1** and the particular alpha-numeric string as arguments for the SetString( ) call **416***c*. A PID that maps an alpha-numeric string to name of a video segment is also included. This information is similarly loaded into the corresponding traffic flow data structure **234** by the flow declaration component **226**. A generic Set( ) call **416***d* may be used for specifying traffic flow characteristics that do not correspond to integer, floating decimal point or alpha-numeric string formats. For example, program **224** may specify a policy element in the well-known eXternal Data Representation (XDR) format. This XDR policy element is included as an argument in the Set( ) call **416***d* to the flow declaration component **226**, which, in response, simply copies the XDR policy element into traffic flow data structure **234**. The policy element may alternatively be specified using the well-known Abstract Syntax Notation One (ASN.1) format, or any other similar translation or encoding techniques.

The application-level parameters may encompass a whole range of information relating to different aspects of the traffic flow from the application program **224**. For example, application-level parameters include such information as user name (e.g., John Smith), user department (e.g., engineering, accounting, marketing, etc.), application name (e.g., SAP R/3, PeopleSoft, etc.), application module (e.g., SAP R/3 accounting form, SAP R/3 order entry form, etc.), transaction type (e.g., print), sub-transaction type (e.g., print on HP Laser Jet Printer), transaction name (e.g., print monthly sales report), sub-transaction name (e.g., print monthly sales report on A4 paper), application state (e.g., normal mode, critical mode, primary mode, back-up mode, etc.). For a video streaming application, the application-level parameters might include user name, film name, film compression method, film priority, optimal bandwidth, etc. Similarly, for a voice over IP application, the application-level parameters may include calling party, called party, compression method, service level of calling party (e.g., gold, silver, bronze), etc. In addition, for World Wide Web (WWW) server-type applications, the application-level parameters may include Uniform Resource Locator (URL) (e.g., http://www.altavista.com/cgi-in/query?pg=aq&kl=en&r=&search=Search&q=Speech+ne ar+recognition), front-end URL (e.g., http:/Hwww.altavista.com), back-end URL (e.g., query?pg=aq&kl=en&r=&search=Search&q=Speech+near+recognition), mime type (e.g., text file, image file, language, etc.), file size, etc. Those skilled in the art will recognize that many other application-level parameters may be defined.

Application program **224** can also retrieve information stored at the traffic flow is data structure **234** by issuing one or more Get API system calls **418** (FIG. **4B**). For example, program **224** may issue a GetSourcePort( ) call **418***a* using the returned bindings handle **H1** as an argument. In response, flow declaration component **226** parses the traffic flow data structure **234** and retrieves the source port information stored therein. Component **226** then returns the GetSourcePort( ) call **418***a* to program **224** with the source port as an argument. Program **224** may issue similar Get API calls to retrieve other network and transport layer parameters stored at the traffic flow data structure **234**.

It should be understood that additional "Get" API system calls may be defined for retrieving application-level information from the traffic flow data structure **234**.

After issuing the application-level Set API calls **416**, if any, the corresponding traffic flow data structure **234** is complete. That is, data structure **234** has been loaded with each of the identifying characteristics specified by the application program **224** for the anticipated traffic flow.

In accordance with the invention, the flow declaration component **226** also opens a communication session with the local policy enforcer **210** and exchanges one or more Application Parameters Declaration (APD) messages. In the preferred embodiment, the flow declaration component **226** opens a reliable, connection-based "socket" session using the well-know Transport Control Protocol (TCP) protocol of the TCP/IP communication protocol stack. A "socket" is essentially an interface between the application and transport layers of a communication protocol stack that enables

the transport layer to identify which process it must communicate with in the application layer. A socket interfaces to a TCP/IP communication protocol stack via APIs consisting of a set of entry points into the stack. Applications that require TCP/IP connectivity thus use the socket APIs to interface into the TCP/IP stack. For a connection-oriented protocol (such a TCP), the socket may be considered a "session".

It should be understood that other protocols, including but not limited to connectionless protocols such as UDP, may be used to establish communication between the flow declaration component **226** and the local policy enforcer **210**. Additionally, component **226** may communicate with local policy enforcer **210** at the network layer by addressing IP format APD messages to end station **212** (i.e., using the same destination address as the anticipated traffic flow) with the well-known Router Alert IP option asserted. Here, local policy enforcer **210** will intercept such asserted network layer packets and may act on them itself and/or forward them to some other network device.

Component **226** may be preconfigured with the IP address of the local policy enforcer **210** or it may dynamically obtain the address of a local policy enforcer. For example, component **226** or application program **224** may broadcast an advertisement seeking the IP address of an intermediate network device that is capable of obtaining and applying policy or service treatments to the anticipated traffic flow from program **224**. Local policy enforcer **210** is preferably configured to respond to such advertisements with its IP address.

Component **226** may receive a "virtual" address that corresponds to a group of available local policy enforcers in a manner similar to the Standby Router Protocol described in U.S. Pat. No. 5,473,599, which is hereby incorporated by reference in its entirety. A single "active" local policy enforcer may be elected from the group to perform the functions described herein.

It should be further understood that the flow declaration component **226** preferably opens one TCP session with the local policy enforcer **210** per application program **224** per network interface card (NIC). More specifically, if host/servet **222** is connected to network **200** through multiple LANs (each with a corresponding NIC), then traffic flows from program **224** may be forwarded onto any of these LANs. To ensure that the appropriate policy or service treatments are applied regardless of which LAN initially carries the flow, flow declaration component **226** preferably establishes a separate communication session with a local policy enforcer **210** through each LAN (i.e., through each NIC) for every program **224** that requests services from component **226**.

In particular, flow declaration component **226** directs message generator **230** to formulate a Client Open message **420** for forwarding to the local policy enforcer **210**. The Client Open message **420** establishes communication between the local policy enforcer **210** and the flow.declaration component **226** and may be used to determine whether the local policy enforcer **210** has the resources to monitor the anticipated flow from the application program **224** and to apply the appropriate policy or service treatments. FIG. **5**A is a block diagram of the preferred format of the Client Open message **420**. In particular, the Client Open message **420** includes at least two elements: a header **510** and a timer area **512**. The header **510** includes a version field **516**, a flags field **518**, an operation code field **520** and a message length field **524**. It may also include one or more unused fields, such as field **522**. Version field **516** preferably contains the version of the software being implemented at the flow declaration component **226**. Flags field **518** preferably contains at least one flag that may be asserted or de-asserted by the flow declaration component **226**, as described below. The operation code field **520** indicates the type of APD message. For a Client Open message **420**, for example, field **520** is preferably loaded with the value "7". The message length field **524** specifies the length (in octets) of the Client Open message **420**.

The timer area **512** includes a length field **526** which specifies the length (preferably in octets) of the timer area **512**, a Class Number (C-Num) field **528**, a Class Type (C-Type) field **530** and a Keep Alive Timer Value field **532**. Timer area **512** may also include one or more unused fields, **534**, **536**. The Class Number field **528** is loaded with an agreed-upon value (e.g., "11") indicating that this portion of the Client Open message **420** (i.e., timer area **512**) contains a keep alive timer value. Where multiple types may exist for a given class number, the Class Type field **530** is used to specify the particular type, Here, field **530** is preferably set to "1". Flow declaration component **226** preferably loads the Keep Alive Timer Value field **532** with a proposed time value (e.g., 30 seconds) to be used for maintaining the TCP session in the absence of substantive APD messages, as described below.

Message generator **230** preferably passes the Client Open message **420** down to the communication facility **228** where it is encapsulated into one or more TCP packets and forwarded to the local policy enforcer **210** in a conventional manner. The APD messages, such as the Client Open message **420**, preferably use a well-known destination port number, such as **1022**. The source destination port for the flow declaration component **226** may be dynamically agreed-upon when the TCP session with the local policy enforcer **210** is first established. At the local policy enforcer **210**, message **420** is received at the communication engine **312** and passed up to the traffic flow state machine engine **310**. The traffic flow state machine engine **310** examines the message **420** which it recognizes as a Client Open message due to the value (e.g., "7") loaded in the operation code field **520**. Local policy enforcer **210** may first determine whether it has adequate resources to accept a new client. For example, local policy enforcer **210** may include an admission control module (not shown) that determines the percentage of time that its central processing unit (CPU) has remained idle recently, its available memory (for storing policies associated with component **226**) and the availability of its traffic management resources, such as meter **322**, marker **324** and shaper/dropper **326**, to manage additional traffic flows.

Assuming local policy enforcer **210** has sufficient available resources, it replies to the flow declaration component **226** with a Client Accept message **422**. The format of the Client Accept message **422** is similar to the format of the Client Open message **422** shown in FIG. **5**A. In particular, the Client Accept message **422** also includes a header that is similar to header **510** and a timer area that is similar to timer area **512**. The operation code for the Client Accept message **422** (which is loaded in field **520**) is another predefined value (e.g., "8") so that flow declaration component **226** will recognize this APD message as a Client Accept message. The traffic flow state machine engine **310** also loads a value in the Keep Alive Timer Value field **532** which may correspond to the value proposed by component **226** or may be a new value selected by the local policy enforcer **210**.

The traffic flow state machine engine **310** hands the Client Accept message **422** to its communication engine **312** which

may encapsulate the message as required and forwards it to the host/server **222**. At the host/server **222** the message is received at the communication facility **228** and passed up to the flow declaration component **226** where it is examined. Flow declaration component **226** examines the operation code field **520** and "learns" that it is a Client Accept message. Flow declaration component **226** also examines the keep alive timer field **532** to determine what value has been specified by local policy enforcer **210**, which is used to generate additional APD messages, as described below.

It should be understood that the flow declaration component **226** may issue the Client Open message **420** as soon as the StartUp( ) call **420** is issued if not earlier.

When application program **224** is ready to begin transmitting the anticipated traffic flow (e.g., the IBM stock quote form) to end station **212**, it issues a BeginFlow( ) call **424***a* to the flow declaration component. Preferably, the BeginFlow( ) call **424***a* is issued slightly before (e.g., 50 ms) program **224** begins forwarding the message to the communication facility **228**. It should be understood, however, that the BeginFlow( ) call **424***a* may be issued at the same time as the anticipated flow to end station **212** is commenced or even slightly later. The application program **224** uses the previously returned handle H1 as an argument to the BeginFlow( ) call **424***a*. If program **224** wishes to receive any feedback regarding the policy or service treatments that are applied to the respective traffic flow, it may also assert a flag argument in the BeginFlow( ) call **424***a* and add one or more callback functions as additional arguments. The callback function preferably identifies an entry point in the application program **224** to which the requested feedback is to be returned. Program **224** may also load other information or data that will simply be returned to it with the requested feedback to assist program **224**, for example, in mapping the returned feedback to a particular task.

The BeginFlow( ) call **424** is received and examined by the flow declaration component **226**, which, in part, determines whether the feedback flag has been set. If so, it also looks for any callback functions and information arguments specified by program **224**. Flow declaration component **226** may also return a flow handle, H2, to program **224** as an argument to the BeginFlow( ) call **424**. Component **226** may also return an acknowledgment or error message as additional arguments. Assuming that the BeginFlow( ) call **424** did not cause any errors, flow declaration component **226** then directs its message generator **230** to formulate a Flow Start APD message **426**.

FIG. **5B** is a block diagram of a preferred Flow Start message **426**, which is similar to the Client Open message **420**. In particular, the Flow Start message **426** includes a header **510** having a flags field **518** and an operation code field **520**, among others. If program **224** requested policy feedback, then message generator **230** preferably asserts the flag in field **518**. In addition, the operation code field **520** is preferably loaded with the value "1" to indicated that this particular APD message is a Flow Start message **426**. Following the header **510** is a handle area **540**, which includes a length field **542** (specifying the length of the handle area **540**), a Class Number (C-Num) field **544**, a Class Type (C-Type) field **546**, a device handle field **548** and a flow handle field **550**. The C-Num field **544** is loaded with an agreed-upon value (e.g., "1") indicating that this portion of the Flow Start message **426** contains a flow handle. The C-Type field **546** may also be set to "1". The device handle field **548** preferably contains a 2 octet identifier selected by the local policy enforcer **210** during establishment of the communication session. For example, the device handle may be "1327". The flow handle field **550** preferably contains the flow handle H2 generated by the flow declaration component **226** in response to the BeginFlow( ) call **424**.

Following the handle area **540** are a plurality of policy bindings **552**, such as policy bindings **552***a*, **552***b* and **552***c*. The policy bindings **552** contain encoded versions of the information stored in the traffic flow data structure **234** that corresponds to the flow handle specified in field **550**. Each policy binding **552**, moreover, has two elements, a policy identifier element **554** and an encoded policy instance element **556**. Basically, the policy identifier element **554** identifies the type or instance of policy element that is contained in the associated encoded policy instance element **556**. Each policy identifier element **554** includes a plurality of fields, including a length field **558** (specifying its length), a policy identifier (Policy ID) type field **560** and a policy identifier field **562**. Each encoded policy instance element **556** similarly includes a plurality of fields, including a length field **564** (specifying its length), an encapsulation type field **566** and an encoded policy element field **568**.

The first policy binding **552***a*, for example, may contain an encoded copy of the source port identified by program **224** with the SetSourcePort( ) call **414***a* and stored at the respective traffic flow data structure **234**. More specifically, message generator **230** loads policy identifier field **562***a* with the type or instance of the policy element (e.g., "source port"). In the preferred embodiment, this name is a Policy Identifier (PID) as specified in the Internet Engineering Task Force (IETF) draft document *COPS Usage for Differentiated Services* submitted by the Network Working Group, dated December 1998, and incorporated herein by reference in its entirety. A PID specifies a particular policy class (e.g., a type of policy data item) or policy instance (e.g., a particular instance of a given policy class) in a hierarchical arrangement. The Policy ID type field **560***a* contains a predefined value reflecting that field **562***a* contains information in PID format. Component **226** preferably includes a Policy Information Base (PIB) for use in deriving the particular policy identifiers, as described in *COPS Usage for Differentiated Services*.

The message generator **230** then accesses the source port information from the respective traffic flow data structure **234** and translates it into a machine independent format suitable for transmission across network **200**. For example, the source port information may be translated in accordance with the ASN.1 translation technique. The encapsulated version of the source port is then loaded in the encoded policy element field **568***a* of binding **552***a*. The encapsulation type field **566***a* contains a predefined value reflecting that the information in field **568***a* has been encapsulated according to ASN.1. Message generator **230** similarly builds additional bindings **552** that contain encapsulated versions of the source IP address, transport protocol, destination port number and destination IP address as specified by program **224** in API calls **414***b*–**414***e* and stored at traffic flow data structure **234**. Message generator **230** also formulates separate bindings **552** for each of the application-level data items established by the application program **224** through application-level API calls **416**. Again, each of these application-level data items may be identified by a corresponding PID which is loaded in the Policy ID type field **562** of the respective binding **552**. The application-level data item is then translated into a machine-independent format (e.g., through ASN.1) and loaded in the respective encoded policy element field **568**, as described above.

It should be understood that other translation techniques, such as XDR, may also be used. It should be further

understood that the contents of other fields, including policy identifier field **556**, should be similarly translated into machine-independent format.

The Flow Start message **426** is then handed down to the communication facility **228** for transmission to the local policy enforcer **210**. At the local policy enforcer **210**, the message **426** is captured by the communication engine **312** and handed to the traffic flow state machine engine **310** which parses the operation code field **520** to determine that the message is a Flow Start APD message. In response, the local policy enforcer **210** proceeds to obtain the particular policy rules or service treatments that are to be applied to this flow (e.g., a stock quote form for IBM). In particular, the local policy enforcer **210** formulates a Request Policy message **428** for transmission to the policy server **216**. In the preferred embodiment, the format of the Request Policy message **428** corresponds to the Request message of the Common Open Policy Service (COPS) Protocol specified in the IETF draft document *The Common Open Policy Service (COPS) Protocol*, dated Aug. 6, 1998, and incorporated herein by reference in its entirety.

According to the COPS protocol, Request messages include a plurality of flags, such as a request type flag and a message flag, and a plurality of objects. The request type flag for message **428** is preferably set to the COPS value that corresponds to "Incoming-Message/Admission Control Request" type COPS messages and the message type flag should be set to "1". Furthermore, the "In-Interface" object of the Request Policy message **428** is preferably set to the VLAN designation associated with the local policy enforcer's interface at which the Flow Start message **426** was received. The bindings **552** of the Flow Start message **426**, which may not be meaningful to the local policy enforcer **210**, are preferably loaded (i.e., copied as opaque objects) into the Client Specific Information (ClientSI) object portion of the Request Policy message **428**. The local policy enforcer **210** also loads a unique handle that identifies the anticipated traffic flow from program **224** into the Request Policy message **428**. This handle, moreover, is used in all messages exchanged between the local policy enforcer **210** and the policy server **216** for this anticipated traffic flow. The handle may be the flow handle H2 previously returned by the flow declaration component **226**.

It should be understood that intermediate network devices, such as local policy enforcer **210**, may learn of the identity of the policy server **216** through any conventional means, such as manual configuration or a device configuration protocol.

The Request Policy message **428** is received at the policy server **216**, which examines the network parameters specified for the anticipated traffic flow, including the IP addresses, port numbers and transport protocol. The policy server **216** also examines the application-level parameters specified by program **224** and provided to the policy server **216** in the Request Policy message **428**. Based on this information, the policy server **216** makes a decision regarding the policy rules or service treatments to be applied to this traffic flow. For example, as described in co-pending U.S. patent application Ser. No. 09/179,036, which is hereby incorporated by reference in its entirety, the policy server **216** may obtain information from the repository **218** and/or network administrator via end station **220** and, in response, formulate one or more traffic management rules, such as classification, behavioral or configuration rules. More specifically, server **216** may formulate one or more classification rules for instructing the local policy enforcer **210** to classify data packets and frames from this traffic flow with a given DS codepoint, IP Precedence and/or user priority. Policy server **216** may also formulate one or more behavioral rules that instruct the local policy enforcer **210** to map packets with the given DS codepoint to a particular queue (e.g., **330***d*) and to apply a particular scheduling algorithm (e.g., WFQ). These policy decisions or rules are then loaded into a Policy Decision message **430** and sent from the policy server **216** to the local policy enforcer **210**.

Communication engine **312** captures the Policy Decision message **430** and forwards it to the traffic flow state machine engine **310**, which, in turn, extracts the policy decisions or rules contained in the message **430**. Traffic flow state machine engine **310** preferably establishes a flow state (not shown) for the anticipated traffic flow that includes information identifying the anticipated traffic flow (such as IP addresses, port numbers and transport protocol) and the policy decisions or rules to be applied to that traffic. Traffic flow state machine engine **310** may also build one or more data structures (such as tables) to store the mappings contained in the Policy Decision message **430**.

As packets or frames are received at the local policy enforcer **210**, they are examined by the packet/frame classifier **314**. More specifically, the packet/frame classifier **314** parses the source and destination port fields **152, 154** (FIG. 1C) and the IP source and destination address fields **126, 128** and the protocol field **124** (FIG. 1B). This information is then supplied to the traffic flow state machine engine **310**, which determines whether a traffic flow state has been established for such packets or frames. Assuming the packets or frames correspond to the anticipated flow from the program **224** to end station **212** (e.g., the IBM stock quote form), a traffic flow state will exist and have associated policy rules or service treatments as specified in the Policy Decision message **430** from policy server **216**. Local policy enforcer **210** then applies the specified treatments to these packets or frames. For example, the traffic flow state machine engine **310** may instruct the packet/frame classifier, to set the DS field **132** (FIG. 1B) of such packets or frames to a value associated with best effort traffic. Similarly, the traffic flow state machine engine **310** may instruct the queue selector/mapping entity **318** to place these packets or frames in a particular (e.g., moderate priority) queue. Alternatively or in addition, packet/frame classifier may be instructed to load the ToS field **122** (FIG. 1B) or the user_priority field **108** (FIG. 1A) with predetermined values so as to implement these treatments at other intermediate network devices, such as device **208**.

To the extent the application program **224** requested feedback as to the policy or service treatments applied to this traffic flow, the local policy enforcer **210** may formulate and send one or more Decision Feedback APD messages **432** to the flow declaration component **226**. The Decision Feedback message **432** is similar in format to the Flow Start message **426**. In particular, the Decision Feedback message **432** has a header **510** and a handle area **540**. For Decision Feedback messages **432**, the operation code field **520** is preferably loaded with the value "3". Appended to the handle area **540** are one or more decision bindings (not shown) that are similar in format to the policy bindings **552**. In particular, each decision binding contains a treatment specified by the policy server **216** and applied by the local policy enforcer **210**. For example, a first decision binding may provide that the specified traffic flow is being marked with a particular DS codepoint. Other decision bindings may specify the IP Precedence or user_priority values being entered in fields **122, 108**, respectively, of this traffic flow. Other decision bindings may be more abstract and describe abstract service

classes granted to the traffic flow. The Decision Request message **432** is received at the communication facility **228** and passed up to the flow declaration component **226**. The flow declaration component **228** extracts the particular treatments from the decision bindings and returns them to the application program **224** through a callback function **434** specified by the application program **224** in the BeginFlow( ) call **424**.

In order to maintain the TCP session established between the flow declaration component **226** and the local policy enforcer **210**, the flow declaration component **226** may send one or more Keep Alive APD messages **436**. The Keep Alive message **436** simply includes a header **510** with the operation code field set to "9" and the message length field **524** set to "0". Flow declaration component **226** preferably sends at least one Keep Alive message **436** within every time period specified in the keep alive timer value field **532** of the Client Accept message **422**.

It should be understood that the policy server **216** may unilaterally send a Decision Change message **438** to the local policy enforcer **210** if a change in the previously supplied policy rules or service treatments occurs after the Policy Decision message **430** was sent. For example, the policy server **216** may obtain up-dated information from the repository **218** or from the network administrator through end station **220**. This up-dated information may affect the policy rules or service treatments previously supplied to the local policy enforcer **210**. In response, the policy server **216** preferably formulates and sends the Decision Change message **438**. The format of the Decisions Change message **438** is preferably the same as the Policy Decision message **430**. The Decision Change message **438** is similarly captured at the communication engine **312** of the local policy enforcer **210** and forwarded to the traffic flow state machine engine **310**.

To the extent the Decision Change message **438** includes new policy rules or service treatments, the traffic flow state machine **310** preferably up-dates its traffic flow state accordingly. In addition, the traffic flow state machine **310** applies the up-dated policy rules or service treatments to subsequently received packets or frames that correspond to the traffic flow. The local policy enforcer **210** may also generate and send a Decision Feedback message (like message **432**) to component **226** if feedback was requested by program **224**.

The policy server **216** may also transmit one or more Decision messages to other intermediate network devices, such as device **208**, that are along the path of the anticipated traffic flow from host/server **222** to end station **212**. These Decision messages similarly inform the intermediate network devices as to what policy rules or service treatments to apply to the traffic flow from program **224**, which presumably has already been classified by the local policy enforcer **210**. Policy server **216** is thus able to provide end-to-end quality of service support.

It should be understood that the local policy enforcer **210** and the policy server **216** may exchange additional COPS messages as required, such as COPS Client Open and COPS Client Accept messages among others.

The local policy enforcer **210** may also send one or more Keep Alive APD messages **440** to the flow declaration component **226** at the host/server **222**. The Keep Alive message **440** from the local policy enforcer **210** preferably has the same format as Keep Alive message **436** from component **226**.

It should be further understood that the application program **224** may change certain characteristics associated with the traffic flow if the nature of the flow changes over time. For example, after reviewing the quote for IBM stock, the user at end station **212** may decide to place a "buy" order for IBM stock. In response, application program **224** may transmit a stock transaction form. Furthermore, the policies or service treatments to be applied to the traffic flow corresponding to the stock quote form may be very different from the treatments that should be applied to the traffic flow corresponding to the stock transaction form. Accordingly, the program **224** may issue one or more new application-level Set API calls **442**. For example, the program may issue a SetInteger( ) call **442***a*, a SetString( ) call **442***b*, a SetFloat( ) call **442***c* and/or a Set( ) call **442***d*. These calls are generally the same as the previously described application-level Set API calls **416** and, although the program **224** utilizes the previously returned handle H1 as an argument, it enters new or updated information (e.g., stock transaction versus stock quote forms). In response, the flow declaration component **226** overwrites the corresponding entries in the respective traffic flow data structure **234** with the new or up-dated information.

The application program **224** then issues a BeginUpdatedFlow( ) call **444** at or about the time that it begins forwarding the stock transaction form to the user at end station **212**. The BeginUpdatedFlow( ) call **444** is preferably the same as the BeginFlow call **424** described above. In response, the flow declaration component **226** directs the message generator **230** to generate and send a Flow Update APD message **446** to the local policy enforcer **210**. The Flow Update message **446** is similar to the Flow Start message **424** and also includes one or more bindings generated from the information stored in the respective traffic flow data structure **234**. Since the information contained in the traffic flow data structure **234** has been up-dated (through the issuance of the Set API calls **442**), the bindings will be different from the bindings appended to the original Flow Start message **426**.

At the local policy enforcer **210**, the Flow Update message **446** is examined and a Request Policy Update message **428** is preferably formulated and sent to the policy server **216**. The Request Policy Update message **428** has the same general format as the original COPS Request Policy message **448**, although it includes the new bindings generated as a result of the Set API calls **442**. The policy server **216** examines the Request Policy Update message **448** and, in response, obtains the appropriate policy rules or service treatments for this up-dated traffic flow. The policy server **216** then loads these up-dated policy rules or service treatments in a Policy Decision Update message **450**, which is sent to the local policy enforcer **210**. Since at least some of the traffic characteristics have changed, the policies or treatments contained in the Policy Decision Update message **450** may be different than the treatments previously provided in the Policy Decision **430**. For example, the up-dated policies may provide that this traffic flow is to be classified as high priority and granted excellent effort treatment. Simnilarly, the up-dated policies may provide that the DS field **132** of packets or frames from this traffic flow should be loaded with a DS codepoint associated with expedited forwarding.

The Policy Decision Update message **450** is received at the local policy enforcer **210** which modifies the corresponding traffic flow state with the up-dated policies. The local policy enforcer **210** also applies these up-dated policies to any subsequently received packets or frames from the host/server **222** that satisfy the previously identified network and transport layer parameters (e.g., IP addresses, port numbers

and transport protocol). Local policy enforcer **210** may also provide feedback to component **226** as described above.

When the traffic flow between the application program **224** and end station **212** is finished, program **224** preferably issues a ReleaseFlow( ) call **452** to the flow declaration component **226** using the previously returned flow handle H2 as an argument. Flow declaration component **226** may return an acknowledgment or an error message to the program **224**. In response, the flow declaration component **226** directs message generator **230** to formulate a Flow End APD message **454**. The format of the Flow End message **454** is preferably the same as the Flow Start message **426**, although the operation code field **520** is preferably loaded with "2" to signify that it is a Flow End message. Although the flow declaration component **226** forwards the Flow End message **454** to the local policy enforcer **210**, it preferably does not discard the traffic flow data structure **234**.

In response, the local policy enforcer **210** formulates a COPS Request message **546** to inform the policy server **216** that the respective traffic flow is finished. The policy server **216** may reply with a Decision message **458** authorizing the local policy enforcer **210** to erase the traffic flow state which was established for this particular flow. If the application program **224** subsequently initiates another traffic flow with the same end station **212**, it may re-use the information stored in the traffic flow data structure **234** by issuing another BeginFlow( ) call **424** utilizing the previously returned bindings handle H1. The flow declaration component **226**, in response, proceeds as described above by sending a Flow Start message **426** to the local policy enforcer **210**.

The application program **224** may also issue a DestroyBindings( ) call **460** to the flow declaration component **226** whenever it concludes that the bindings are no longer needed. Program **224** preferably utilizes the previously returned bindings handle H1 as an argument to the DestroyBindings( ) call **460**. In response, component **226** preferably discards the contents of the traffic flow data structure **234** that corresponds to bindings handle H1.

When the application program **224** is closed it should shutdown all outstanding traffic flow services by issuing corresponding ReleaseFlow( ) calls **452** and it should also destroy all bindings that it created by issuing DestroyBindings( ) calls **460**. In response, component **226** directs message generator **230** to formulate a Client Close APD message **462**. The Client Close message **462** is simply a header **510** with the operation code field **520** loaded with the value "10". In response, the local policy enforcer **210** formulates and sends a COPS Request message **464** to the policy server **216** indicating that the program **224** is closed. The policy server **216** may reply with a COPS Decision message **466** instructing the local policy enforcer **210** to release all of the corresponding traffic flow states that were previously established for the application program **224**.

One skilled in the art will recognize that two or more of the previously described API system calls may be combined into a single call or that any one call may be broken down into multiple calls. One skilled in the art will also recognize that the particular names of the API system calls is unimportant. Thus, it is an object of the present invention to cover the foregoing communicating relation between the application program **224** and the flow declaration component **226**, regardless of the particular implementation ultimately chosen.

It should also be understood that any set of values may be inserted in the operation code field **520** of the APD messages provided that each APD message type (e.g., Client Open, Client Accept, Flow Start, etc.) has a different value assigned to it. Furthermore, if a local policy enforcer is unable to handle a particular application program or traffic flow (e.g., insufficient memory or other resources), it preferably responds to the Client Open message with a Client Close message, rather than a Client Accept message.

In the preferred embodiment, the flow declaration component **226** is implemented in software as a series of steps executed at the host/server **222**. Nonetheless, it should be understood that the method may be implemented, either wholly or in part, through one or more computer hardware devices. Additionally, the present invention is preferably utilized only with traffic flows of sufficient length (e.g., greater than 5–10 packets). The application program **224** may be configured not to request bindings or issue API calls for short traffic flows.

It should be understood that some or all of the above described functionality of the local policy enforcer **210** may be located at the host/server **222**. For example, the host/server **222** may include a traffic flow state machine engine **310** that is capable of sending and receiving COPS Request and Decision messages directly to and from the policy server **216**. In this case, the Client Open, Flow Start and Flow Update messages are simply inter-process communications within the host/server **222**, rather than being forwarded across the network. The operating system at the host/server **222** may also include one or more resources that may be utilized to provide traffic management services, such as classifying packets and frames (e.g., loading the DS field **132**, ToS field **122** and/or user priority field **108**); scheduling packet and frame forwarding from different priority queues; etc.

It should be further understood that the local policy enforcer **210** may make policy or service treatment decisions for traffic flows identified by the flow declaration component **226** without querying the policy server **216**. That is, the local policy enforcer **210** may cache certain policy rules or treatments.

In another aspect of the invention, the application program **224** may request policy decisions in advance of issuing the BeginFlow( ) call **424**. For example, program **224** may only have a small number of application-level parameter bindings. After creating the bindings (using only the application-level parameters) as described above, the program **224** may issue a GetFlowDecision( ) system call to component **226** and, in return, receive a handle, H3. Component **226** issues an Obtain Decision APD message to the local policy enforcer **210** for each binding, including the specified application-level parameters. The local policy enforcer **210** will obtain the appropriate policy rules or service treatments to be applied to these, as yet un-specified, "flows" as described above.

When program **224** is about to begin a flow corresponding to one of these bindings, it may issue a BeginFlow( ) call, including the network and transport layer parameters for the traffic flow and the handle H3 for the corresponding application-level bindings. Component **226** then forwards this information in a Flow Start message **426** to the local policy enforcer **210** as described above. Since the local policy enforcer **210** has already obtained the policy or service treatments to be applied to this flow, it need not query the policy server **216**. Instead, the local policy enforcer **210** simply monitors the traffic from host/server **222** and, when it identifies the specified traffic flow, applies the previously received policy rules or service treatments.

Enhanced RSVP Messaging

In a further aspect of the invention, the flow declaration component **226** may be configured to exchange one or more

modified Resource reSerVation Protocol (RSVP) messages with the local policy enforcer **210** in place of the APD messages described above. RSVP is a well-known Internet Control protocol for reserving resources, typically bandwidth, between a sender entity and a receiver entity. RSVP is defined at Request for Comments (RFC) **2205**, September 1997, from the Network Working Group of the IETF, and is hereby incorporated by reference in its entirety. The protocol defines two fundamental message types: RSVP path messages (Path) and reservation request messages (Resv). Basically, senders transmit Path message downstream throughout the network to potential receivers offering to supply a given message stream. Receivers, wishing to obtain the proposed message stream, transmit Resv messages that are propagated upstream all the way back to the sender. At each intermediate node in the network, bandwidth resources are reserved to ensure that the receiver will obtain the message stream.

In this embodiment of the present invention, component **226**, rather than generating and forwarding the Flow Start APD message **426** in response to the BeginFlow( ) call **424**, formulates and sends a modified RSVP Path message to the local policy enforcer **210**. FIG. **6** is a block diagram illustrating the preferred format of a modified RSVP Path message **610**. Modified Path message **610** carries the network and transport layer parameters and application-level parameters specified for the anticipated traffic flow. In particular, message **610** preferably includes at least three elements: an RSVP header **612**, a first area **614** (which carries the network and transport layer parameters) and at least one RSVP Policy-Data object **616** (which carries the application-level parameters). As provided in RFC 2205, the RSVP header includes a version field **618**, a flags field **620**, a message type field **622**, an RSVP checksum field **624**, a Send Time To Live (TTL) field **626**, a reserved field **628** and an RSVP length field **630**.

Component **226** preferably loads version field **618**, which corresponds to the version of RSVP, with the appropriate value (e.g., "1"). Flags field **620** is preferably de-asserted as no flags are presently defined. Message type field **622**, which indicates the type of message (e.g., "1" for RSVP Path messages and "2" for RSVP Resv messages) is preferably loaded with the value "1" to indicate that message **610** is a Path message. It should be understood that field **622** may alternatively be loaded with a new value to indicate that message **610** is a modified RSVP Path message. The RSVP Checksum field **624** may be loaded with a computed checksum for message **610**. The Send_TTL field **626** is preferably loaded with an IP time to live value, and the RSVP length field **630** preferably contains the length of message **610**.

The first area **614** preferably includes an RSVP sender template object **632** and an RSVP session object **634**, each having a plurality of fields. More specifically, the sender template and session objects **632**, **634** each have a length field **638** (loaded with the length of the respective object), a class number field (C-Num) **634** and a class type (C-type) field **642**. For the sender template object **632**, which further includes an IP source address (SA) field **644**, a source port number field **646** and may include one or more un-used fields **648**, the respective C-Num field **640** is preferably loaded with "11" to signify that it is an RSVP sender template object and the respective C-Type field **642** may be loaded with "1" to indicate that fields **644** and **646** carry the IPv4 address and the TCP/UDP port number, respectively, at host/server **222** for the anticipated traffic flow. For the session object **634**, which further includes an IP destination address (DA) field **650**, a transport protocol field **652**, a flags field **654** and a destination port number field **656**, the respective C-Num field **640** is loaded with "1" to signify that it is an RSVP session object and the respective C-Type field **642** may be loaded with "1" to indicate that fields **650** and **656** carry the IPv4 address and the TCP/UDP port number, respectively, for the corresponding process at end station **212** for the anticipated traffic flow. Component **226** may assert flags field **654** if it is capable of policing its own traffic flows.

One skilled in the art will recognize that first area **614** of modified RSVP Path message **610** may be modified in any number of ways, including fewer or additional fields or to carry IPv6 information.

The RSVP Policy_Data object **616** also has a length field **638**, a C-Num field **640** and a C-Type **642** field. In addition RSVP Policy_Data object **616** includes a policy-data object field **658**. The respective length field **638** carries the length of object **616** and the respective C-Num field is loaded with "14" to indicate that field **658** is a policy_data object field. The C-Type field **642** of object **616** is preferably loaded with a new value (e.g., "2") to signify that policy_data object field **658** carries application-level parameters. Furthermore, policy_data object field **658** is loaded by component **226** with the application-level bindings specified by program **224** preferably in the manner as described above with reference to FIG. **5B**.

One skilled in the art will also recognize that the application-level parameters may be carried in multiple RSVP Policy_Data objects **616**.

This modified RSVP path message **610** is preferably handed to the communication facility **228** for forwarding to the local policy enforcer **210** where it is examined. In response, the local policy enforcer **210** and the policy server **216** exchange Request Policy **428** and Policy Decision **430** messages, as described above, in order to obtain the policy rules or service treatments to be applied to the traffic flow identified in the modified RSVP Path message **610**. Local policy enforcer **210** also extracts and stores the network and transport layer parameters from the RSVP Sender Template object **614** in order to identify the particular traffic flow from host/server **222**.

The local policy enforcer **210** may also reply to component **226** with a modified RSVP Resv message rather than the Decision Feedback message **432**. This modified RSVP Resv message preferably includes a header similar to header **612**, but with the message type field **622** loaded with the value "2" to indicate that it is an RSVP Resv messages or with a new value to indicate that it is a modified RSVP Resv message. The modified RSVP Resv message also includes one or more RSVP Policy_Data objects similar to object **616**. In this case, however, object **616** carries the decision bindings for the anticipated traffic flow as described above. Component **226** may extract these decision bindings in order to provide feedback to application **224**.

As shown, component **226** utilizes a modified RSVP path message **610** to identify network and transport layer parameters and application-level parameters to the local policy enforcer **210**. The modified RSVP Path message **610**, moreover, is preferably not forwarded by the local policy enforcer **210**, unlike conventional RSVP Path and Resv messages which are propagated all the way between the sender and receiver entities.

It should be understood that the local policy enforcer **210** is preferably in close proximity to host/server **222** so that the classification of packets or frames from the anticipated traffic flow occurs early in their journey through the network

200. It should also be understood that the traffic flow from end station 212 to host/server 222 may similarly be identified and appropriate policy rules or service treatments applied thereto. It should be further understood that the flow declaration component 226 is configured to handle and separately identify multiple traffic flows from multiple application programs executing at the host/server 222 so that the appropriate policy rules or service treatments may be individually applied to each such traffic flow through the network 200. For example, program 224 may be simultaneously sending a print transaction t the print server 214.

The foregoing description has been directed to specific embodiments of the invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. For example, other client-server communications protocols, besides COPS, may be utilized by the policy server and the local policy enforcer. In addition, the present invention may also be utilized with other network layer protocols, such as IPv6, whose addresses are 128 bits long. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A method for applying a service treatment to a plurality of network messages issued by a network entity connected to a computer network, the network messages corresponding to a traffic flow, the computer network configured to support transport and network communication layers and having a policy enforcer, the method comprising the steps of:

receiving from the policy enforcer a request policy message identifying the traffic flow and including one or more application-level parameters;

utilizing at least some of the included application-level parameters to select one or more service treatments to be applied to the traffic flow;

generating a policy decision message containing the one or more service treatments selected for the traffic flow; and

sending the policy decision message to the policy enforcer.

2. The method of claim 1 wherein the application-level parameters specify one or more of the following characteristics: the size of a file being transmitted, a video segment name, a video segment viewer, a user name, a user department, an application identifier, a transaction type, a transaction name, an application state, a calling party, a called party, a compression method, a service level, a uniform resource locator (URL) and a mime type.

3. The method of claim 2 wherein the one or more selected service treatments specified in the policy decision message includes instructions for marking network messages corresponding to the traffic flow with one or more of a selected Differentiated Services Codepoint (DSCP), a selected Type of Service (ToS), and a selected user_priority.

4. The method of claim 1 further comprising the steps of:

formulating one or more classification rules for instructing the policy enforcer to mark network messages corresponding to the traffic flow with one or more of a selected Differentiated Services Codepoint (DSCP), a selected Type of Service (ToS), and a selected user_priority; and

loading the policy decision message with the one or more classification rules.

5. The method of claim 4 wherein the policy enforcer has a plurality of queues and queue scheduling algorithms, the method further comprising the steps of:

formulating one or more behavioral rules for instructing the policy enforcer to map network messages marked with a selected DSCP, ToS and/or user_priority to a particular queue and to apply a designated queue scheduling algorithm.

6. The method of claim 5 wherein a behavioral rule instructs the policy enforcer to apply a Weighted Fair Queuing (WFQ) queue scheduling algorithm.

7. The method of claim 1 wherein the computer network further has a repository configured to store information, and the step of utilizing comprises the steps of:

requesting information from the repository; and

utilizing the information from the repository in selecting the one or more service treatments to be applied to the traffic flow.

8. The method of claim 1 further comprising the step of sending a decision change message to the policy enforcer, following the step of sending the policy decision message, wherein the decision change message contains one or more service treatments that differ from the service treatments specified by the policy decision message.

9. The method of claim 1 further comprising the steps of:

receiving a request policy update message from the policy server, the request policy update message containing traffic flow and/or application-level parameters that differ from the traffic flow and/or application-level parameters contained in the request policy message;

utilizing the traffic flow and/or application-level parameters from the request policy_update message to select one or more new service treatments;

generating a policy decision update message containing the one or more new service treatments; and

sending the policy decision update message to the policy enforcer.

10. A computer readable medium containing executable program instructions for use in applying a service treatment to a plurality of network messages issued by a network entity connected to a computer network, the network messages corresponding to a traffic flow, the computer network configured to support transport and network communication layers and having a policy server, the executable program instructions comprising program instructions for:

receiving from the network entity a message identifying the traffic flow and including one or more application-level parameters;

generating a request policy message for the identified traffic flow, the request policy message containing at least some of the application-level parameters included in the message;

sending the request policy message to the policy server; and

receiving a policy decision message from the policy server specifying one or more service treatments to be applied to the traffic flow, the one or more service treatments based, at least in part, upon the application-level parameters contained in the request policy message.

11. The computer readable medium of claim 10 further comprising program instructions for:

identifying network messages corresponding to the traffic flow; and

applying the one or more service treatments specified in the policy decision message to those network messages identified as corresponding to the traffic flow.

12. The computer readable medium of claim 10 wherein the application-level parameters included in the request

policy message specify one or more of the following characteristics: the size of a file being transmitted, a video segment name, a video segment viewer, a user name, a user department, an application identifier, a transaction type, a transaction name, an application state, a calling party, a called party, a compression method, a service level, a uniform resource locator (URL) and a mime type.

13. The computer readable medium of claim 10 wherein the one or more service treatments specified in the policy decision message include instructions for marking network messages corresponding to the traffic flow with one or more of a selected Differentiated Services Codepoint (DSCP), a selected Type of Service (ToS), and a selected user_priority.

14. The computer readable medium of claim 12 further comprising program instructions for generating and sending to the policy server one or more client accept messages carrying a keep alive timer value.

15. A policy server for use in applying a service treatment to a plurality of network messages issued by a network entity connected to a computer network, the network messages corresponding to a traffic flow, the computer network configured to support transport and network communication layers and having a policy enforcer, the policy server comprising:

means for sending and receiving messages to and from the policy enforcer via the computer network; and

means, responsive to receipt of a request policy message identifying the traffic flow and containing one or more application-level parameters, for selecting one or more service treatments to be applied to the traffic flow, wherein

the one or more service treatments are selected based at least in part on the one or more application-level parameters, and

the message sending and receiving means sends a policy decision message to the policy enforcer carrying the one or more selected service treatments.

16. A policy enforcer for use in applying a service treatment to a plurality of network messages issued by an application program running on a network entity connected to a computer network, the network messages corresponding to a traffic flow, the computer network configured to support transport and network communication layers and having a policy server, the policy enforcer comprising:

means for sending and receiving messages via the computer network to the network entity and the policy server; and

means for applying varying service treatments to traffic flows, wherein

in response to receiving a flow start message from the network entity identifying the traffic flow and containing one or more application-level parameters, the policy enforcer generates and sends to the policy server a request policy message for the identified traffic flow, the request policy message containing at least some of the application-level parameters included in the flow start message,

in response to receiving a policy decision message from the policy server specifying one or more service treatments, applying the one or more service treatments to the traffic flow from the network entity, and

the one or more service treatments based, at least in part, upon the application-level parameters contained in the request policy message.

* * * * *

US006286052B1

(12) **United States Patent**
McCloghrie et al.

(10) **Patent No.: US 6,286,052 B1**
(45) **Date of Patent: Sep. 4, 2001**

(54) **METHOD AND APPARATUS FOR IDENTIFYING NETWORK DATA TRAFFIC FLOWS AND FOR APPLYING QUALITY OF SERVICE TREATMENTS TO THE FLOWS**

(75) Inventors: **Keith McCloghrie**, San Jose, CA (US); **Silvano Gai**, Vigliano d'Asti (IT); **Shai Mohaban**, Sunnyvale, CA (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/206,067**

(22) Filed: **Dec. 4, 1998**

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **709/238**; 709/232; 709/250; 709/236; 370/235
(58) **Field of Search** 709/102, 103, 709/104, 227, 228, 229, 230, 232, 233, 234, 235, 236, 238, 240, 245, 250; 370/229, 230, 235

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,769,810	9/1988	Eckberg, Jr. et al.	370/60
4,769,811	9/1988	Eckberg, Jr. et al.	370/60
5,224,099	6/1993	Corbalis et al.	370/94.2
5,263,157	11/1993	Janis	395/600
5,473,599	12/1995	Li et al.	370/16
5,606,668	2/1997	Shwed	395/200.1
5,666,353	9/1997	Klausmeier et al.	370/230
5,751,967	5/1998	Raab et al.	395/200.58
5,819,042	10/1998	Hansen	395/200.52
5,832,503	11/1998	Malik et al.	707/104
5,842,040 *	11/1998	Hughes et al.	710/11
5,872,928	2/1999	Lewis et al.	395/200.52
5,889,953	3/1999	Thebaut et al.	395/200.51
5,987,513	11/1999	Prithviraj et al.	709/223
6,041,347	3/2000	Harsham et al.	709/220
6,046,980 *	4/2000	Packer	370/230
6,047,322	4/2000	Vaid et al.	709/224
6,091,709 *	7/2000	Harrison et al.	370/235
6,104,700 *	8/2000	Haddock et al.	370/235

OTHER PUBLICATIONS

Ortiz, Jr., S., "Active Networks: The Programmable Pipeline", *Computer* pp. Aug. 19–21, 1998.
IEEE P802.1D Standard (draft 15) "Local and Metropolitan Area Networks", pp. 1, 50–56 and 378–381 (Nov. 1997).

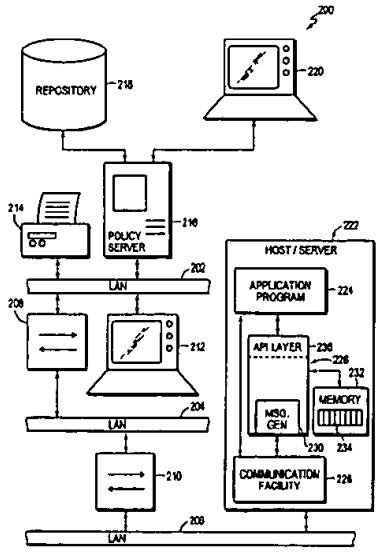
(List continued on next page.)

Primary Examiner— Viet D. Vu
(74) *Attorney, Agent, or Firm*—Cesari & McKenna, LLP

(57) **ABSTRACT**

A system within a computer network identifies specific traffic flows originating from a given network entity and requests and applies appropriate policy rules or service treatments to the traffic flows. A network entity includes a flow declaration component that communicates with one or more application programs executing on the entity. The flow declaration component includes a message generator and an associated memory for storing one or more traffic flow data structures. For a given traffic flow, the application program issues one or more calls to the flow declaration component providing it with information identifying the traffic flows. The flow declaration component then opens a flow management session with a local policy enforcer that obtains policy rules or service treatments for the identified flow from a policy server and applies those rules or treatments to the specific traffic flows from the network entity.

47 Claims, 10 Drawing Sheets



OTHER PUBLICATIONS

"An Emerging Trend in the Internet Services Market", Hewlett–Packard Corp. (date unknown).
Wroclawaski, J., "The Use of RSVP with IETF Integrated Services", IETF Network Working Group (Sep. 1997).
Bernet, Y. et al., "A Framework for Use of RSVP with Diff–serv Networks", IETF (Nov. 1998).
Bernet, Y. et al., "Requirements of Diff–serv Boundary Routers", IETF Differentiated Services (Nov. 1998).
Yadav, S. et al., "Identity Representation for RSVP", IETF (Jan. 1999).
Heinanen, J. et al., "Assured Forwarding PHB Group", IETF (Sep. 1998).
Jacobson, V. et al., "An Expedited Forwarding PHB", IETF Differentiated Services Working Group (Aug. 1998).
Nichols, K. et al., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", IETF Differentiated Services Working Group (Aug. 1998).
Blake, S. et al., "An Architecture For Differentiated Services", IETF Differentiated Services Working Group (Aug. 1998).
Bernet, Y. et al., "A Framework for End–to–End QoS Combining RSVP/Interserv and Differentiated Services", IETF (Mar. 1998).
Yavatkar, R. et al., "A Framework for Policy–based Admission Control", IETF (Nov. 1997).
Boyle, J. et al., "The COPS (Common Open Policy Service)Protocol", IETF (Aug. 1998).
Reichmeyer, F. et al., "COPS Usage for Differentiated Services", IETF Network Working Group (Aug. 1998).
"Cisco IOS® Software Quality of Service Solutions", Cisco Systems, Inc.(Jul. 1998).
"Queuing, Traffic Shaping, and Filtering", Cisco Systems, Inc.(Sep. 1996).
"Network Node Registry Overview" (Jan. 29, 1998).
"Network Node Registry User's Guide" (Apr. 1997).
"Network Node Registry—Access Control Lists" (Apr. 1997).
"Quality of Service Policy Propagation via Border Gateway Protocol", Cisco Systems, Inc. (Feb. 1998).
"Distributed Weighted Random Early Detection", Cisco Systems, Inc., pp. 1–6 (Feb. 1998).
"Distributed Weighted Fair Queuing", Cisco Systems, Inc. (Mar. 1998).
"Action Request System®", Remedy Corporation (1998).
"3COM's Framework for Delivering Policy–Powered Networks", 3Com Corporation (Jun. 1998).

* cited by examiner

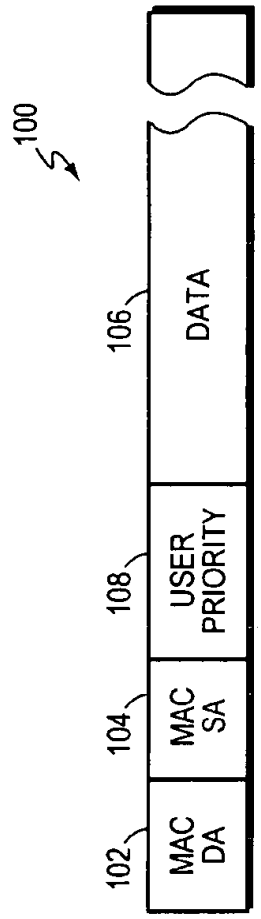


FIG. 1A (PRIOR ART)

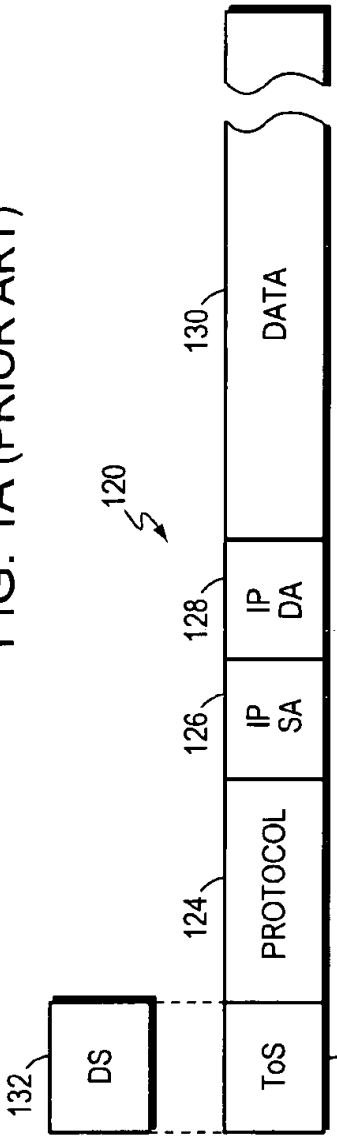


FIG. 1B (PRIOR ART)

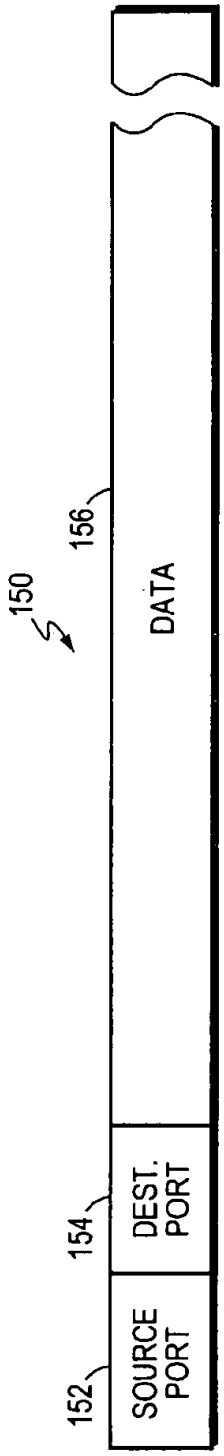


FIG. 1C (PRIOR ART)

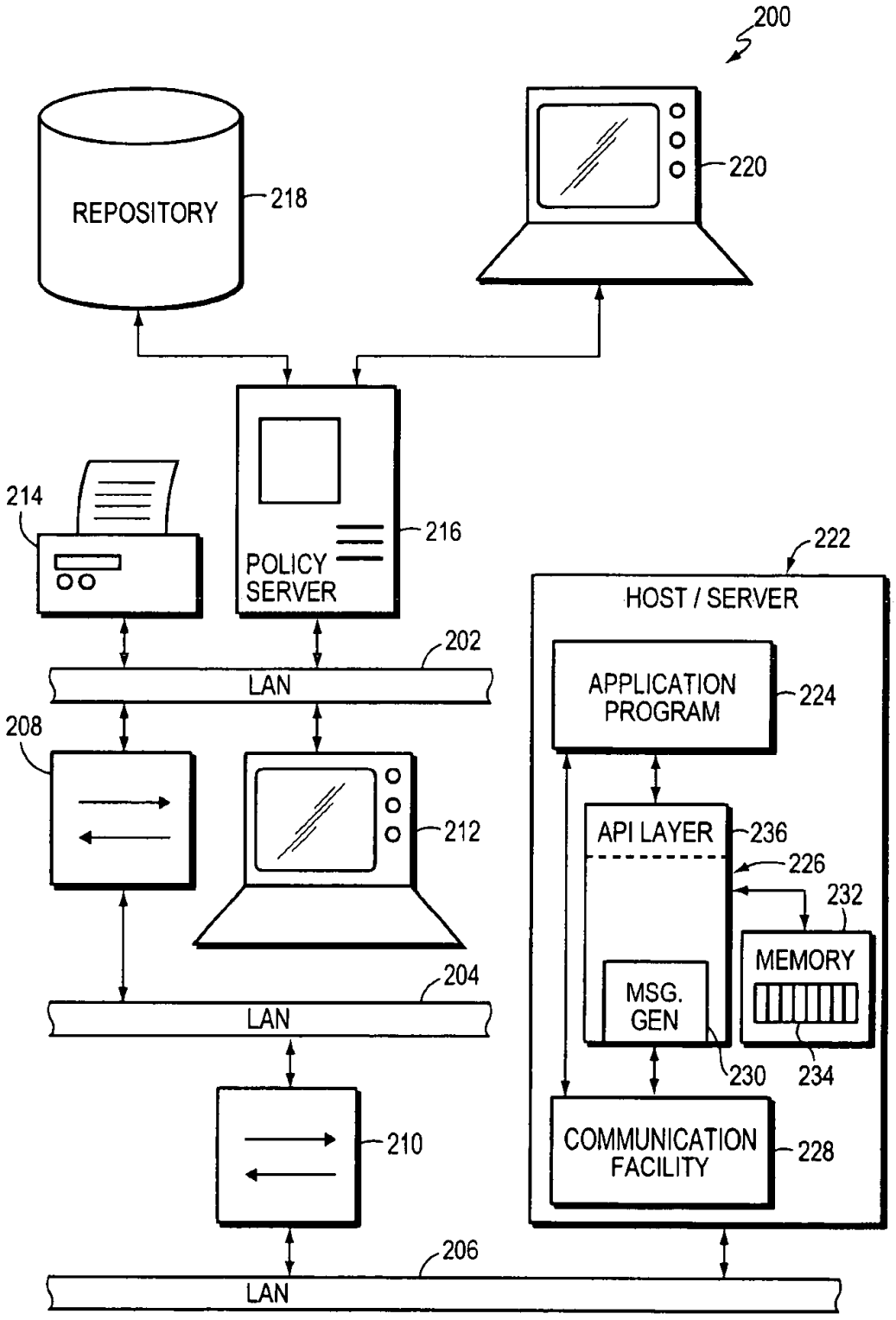

200
REPOSITORY
218
220
214
POLICY SERVER
216
222
HOST / SERVER
202
LAN
APPLICATION PROGRAM
224
208
212
API LAYER
236
226
232
MEMORY
MSG. GEN
230
234
204
LAN
210
COMMUNICATION FACILITY
228
206
LAN

FIG. 2

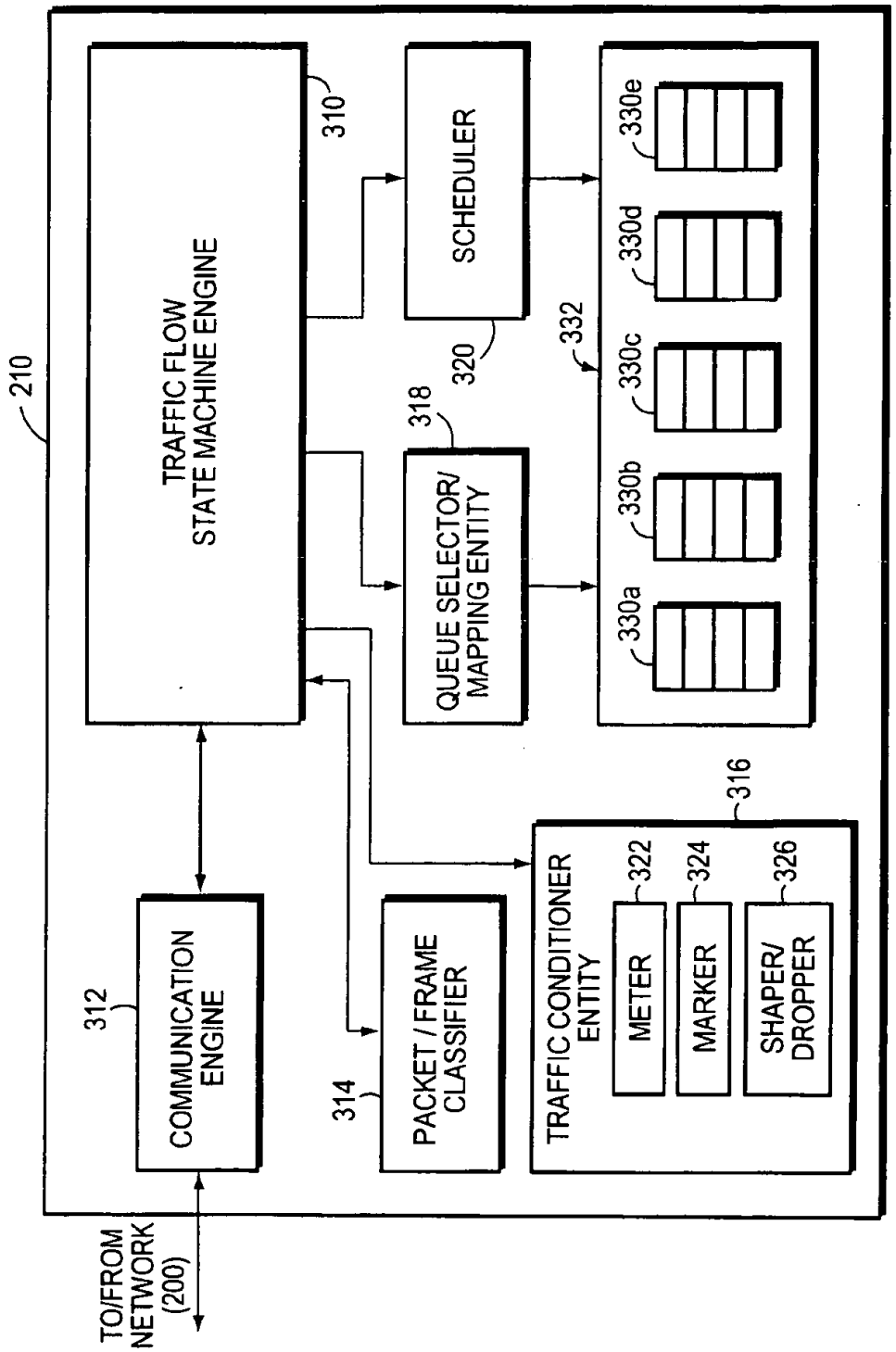

210
TRAFFIC FLOW STATE MACHINE ENGINE
310
SCHEDULER
320
332
330e
330d
330c
330b
330a
318
QUEUE SELECTOR/ MAPPING ENTITY
316
TRAFFIC CONDITIONER ENTITY
322
METER
324
MARKER
326
SHAPER/ DROPPER
312
COMMUNICATION ENGINE
314
PACKET / FRAME CLASSIFIER
TO/FROM NETWORK (200)

FIG. 3

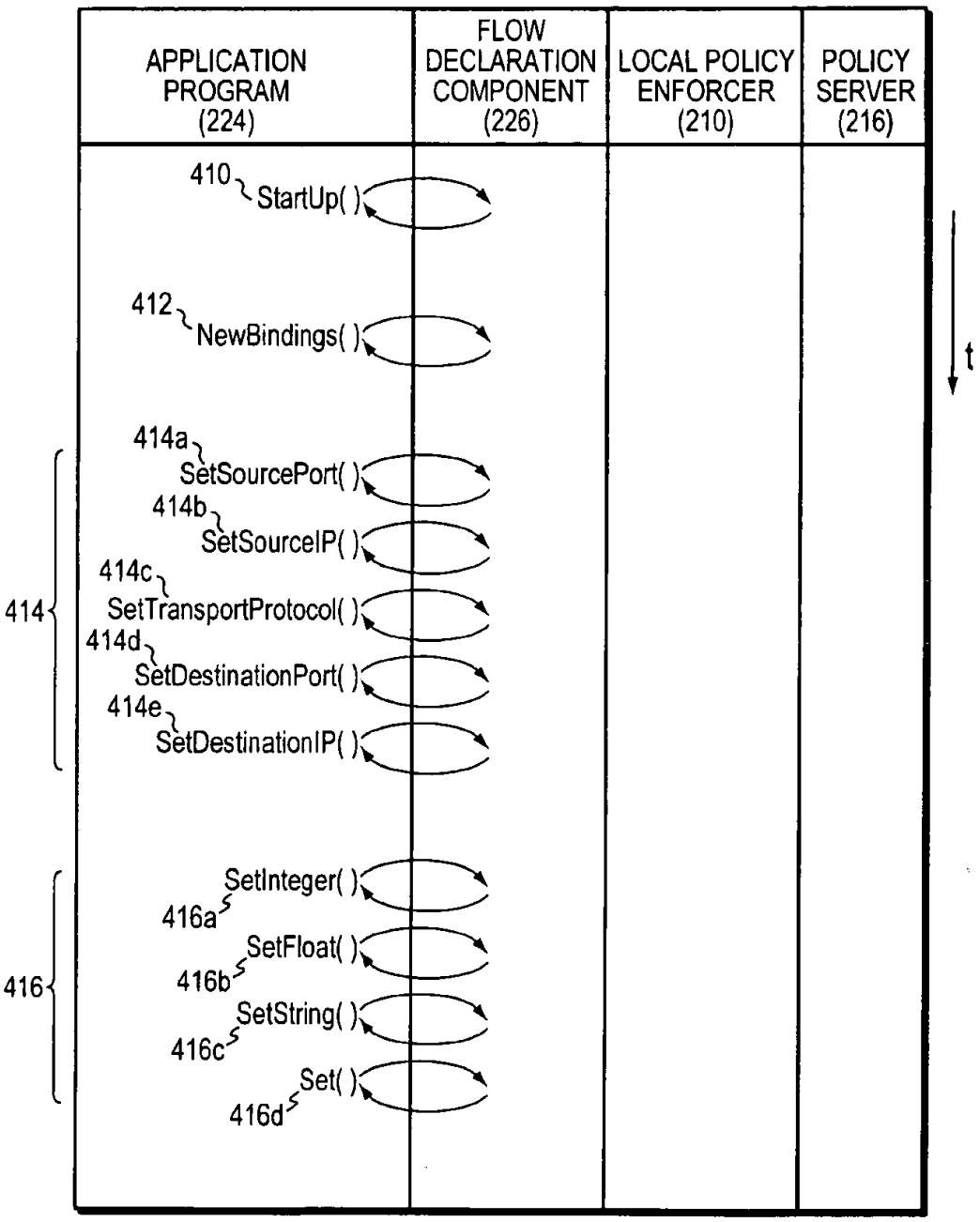

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
410
StartUp( )
412
NewBindings( )
414
414a
SetSourcePort( )
414b
SetSourceIP( )
414c
SetTransportProtocol( )
414d
SetDestinationPort( )
414e
SetDestinationIP( )
416
SetInteger( )
416a
SetFloat( )
416b
SetString( )
416c
Set( )
416d
t

FIG. 4A

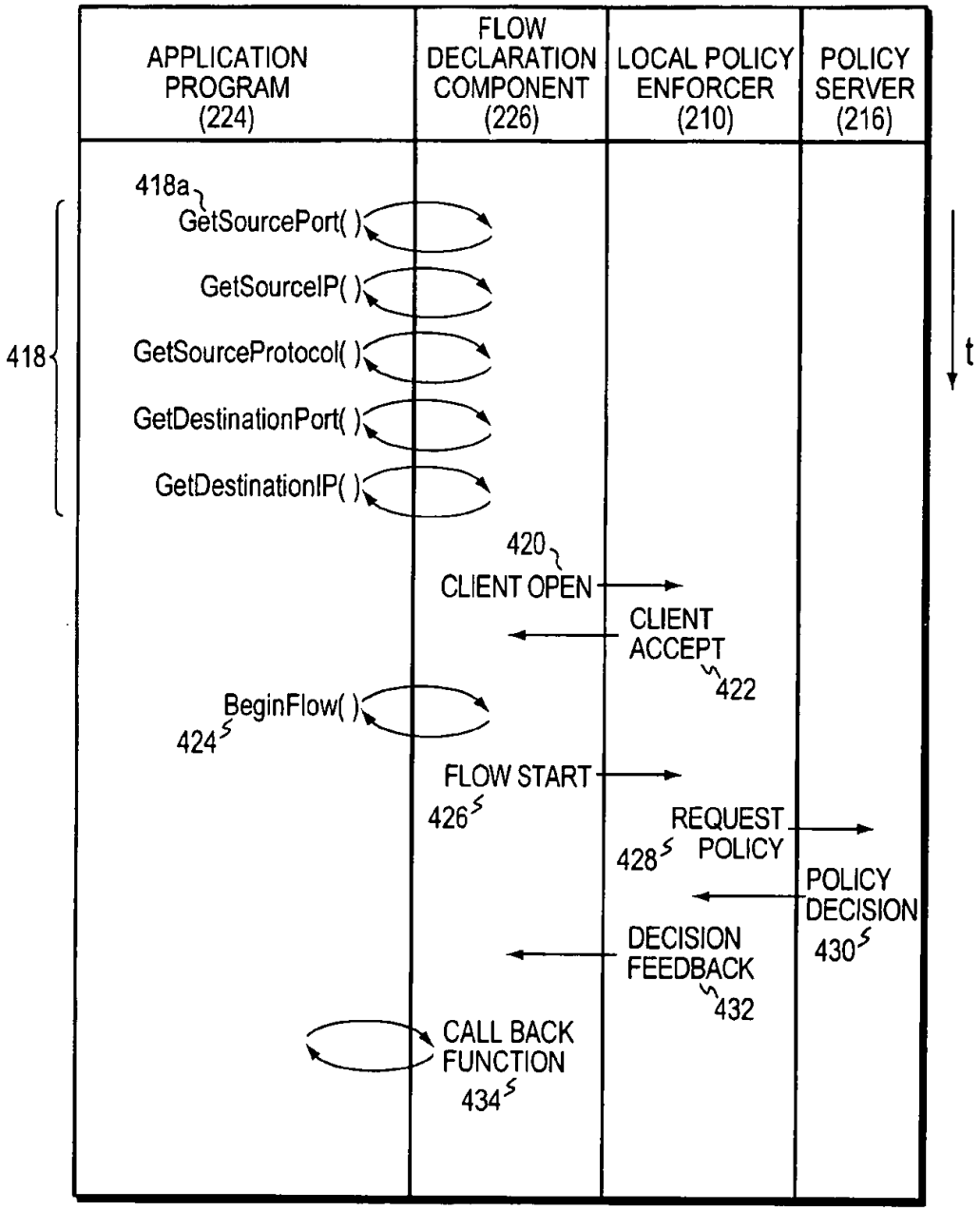

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
418a
GetSourcePort( )
GetSourceIP( )
GetSourceProtocol( )
GetDestinationPort( )
GetDestinationIP( )
418
t
420
CLIENT OPEN
CLIENT ACCEPT
422
BeginFlow( )
424
FLOW START
426
REQUEST POLICY
428
POLICY DECISION
430
DECISION FEEDBACK
432
CALL BACK FUNCTION
434

FIG. 4B

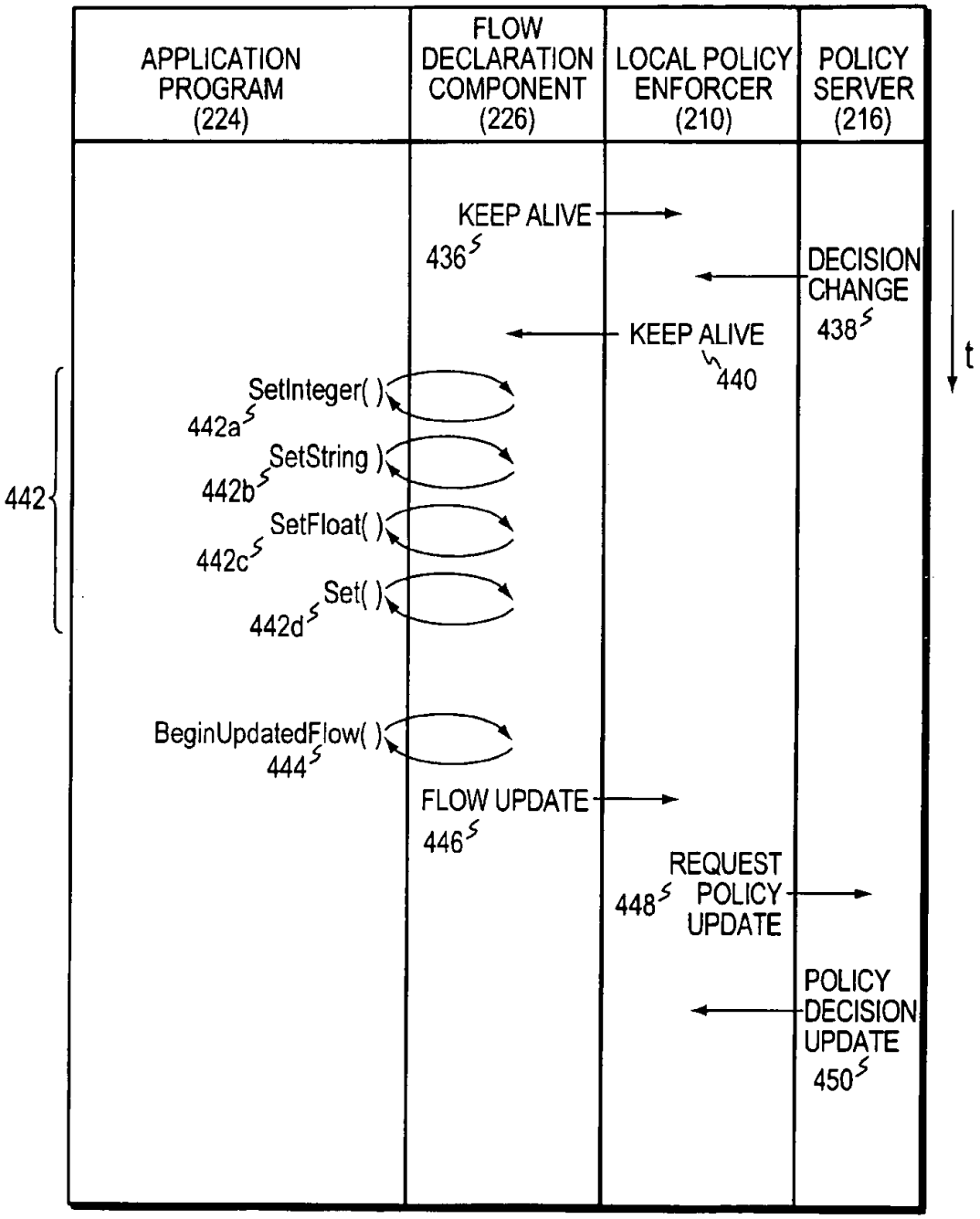

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
KEEP ALIVE
436
DECISION CHANGE
438
KEEP ALIVE
440
t
442
SetInteger( )
442a
SetString)
442b
SetFloat( )
442c
Set( )
442d
BeginUpdatedFlow( )
444
FLOW UPDATE
446
REQUEST POLICY UPDATE
448
POLICY DECISION UPDATE
450

FIG. 4C

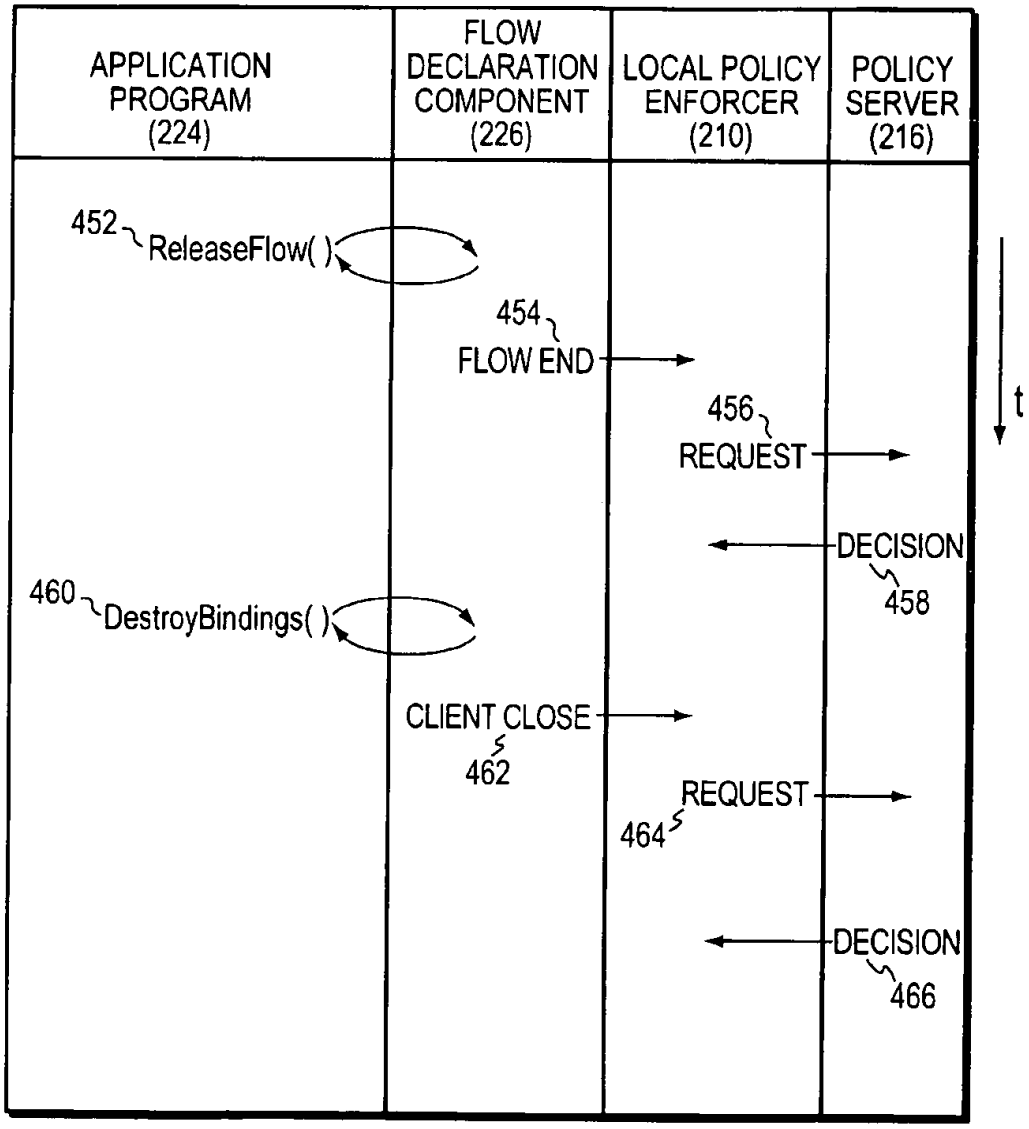

APPLICATION PROGRAM (224)
FLOW DECLARATION COMPONENT (226)
LOCAL POLICY ENFORCER (210)
POLICY SERVER (216)
452
ReleaseFlow( )
454
FLOW END
456
REQUEST
DECISION
458
460
DestroyBindings( )
CLIENT CLOSE
462
REQUEST
464
DECISION
466
t

FIG. 4D

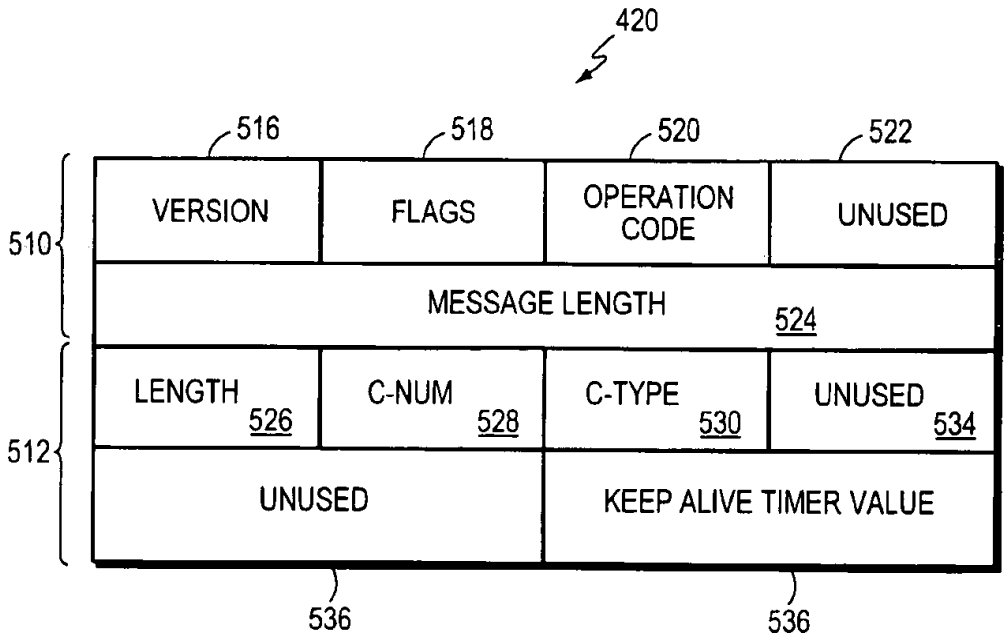

420
516
518
520
522
510
VERSION
FLAGS
OPERATION CODE
UNUSED
MESSAGE LENGTH
524
512
LENGTH
526
C-NUM
528
C-TYPE
530
UNUSED
534
UNUSED
KEEP ALIVE TIMER VALUE
536
536

FIG. 5A

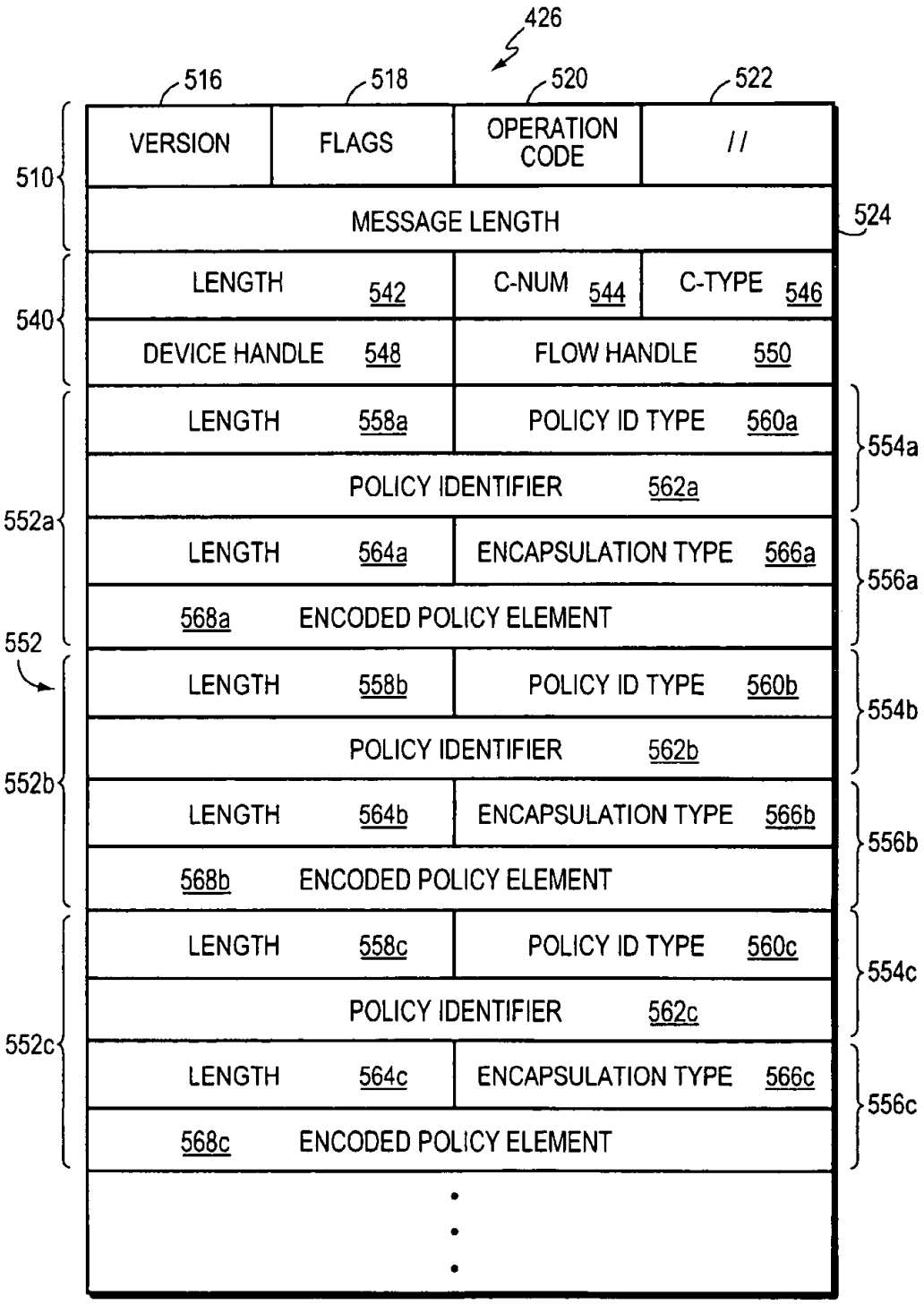

426
516
518
520
522
VERSION
FLAGS
OPERATION CODE
//
510
MESSAGE LENGTH
524
LENGTH 542
C-NUM 544
C-TYPE 546
540
DEVICE HANDLE 548
FLOW HANDLE 550
LENGTH 558a
POLICY ID TYPE 560a
554a
POLICY IDENTIFIER 562a
552a
LENGTH 564a
ENCAPSULATION TYPE 566a
556a
568a ENCODED POLICY ELEMENT
552
LENGTH 558b
POLICY ID TYPE 560b
554b
POLICY IDENTIFIER 562b
552b
LENGTH 564b
ENCAPSULATION TYPE 566b
556b
568b ENCODED POLICY ELEMENT
LENGTH 558c
POLICY ID TYPE 560c
554c
POLICY IDENTIFIER 562c
552c
LENGTH 564c
ENCAPSULATION TYPE 566c
556c
568c ENCODED POLICY ELEMENT

FIG. 5B

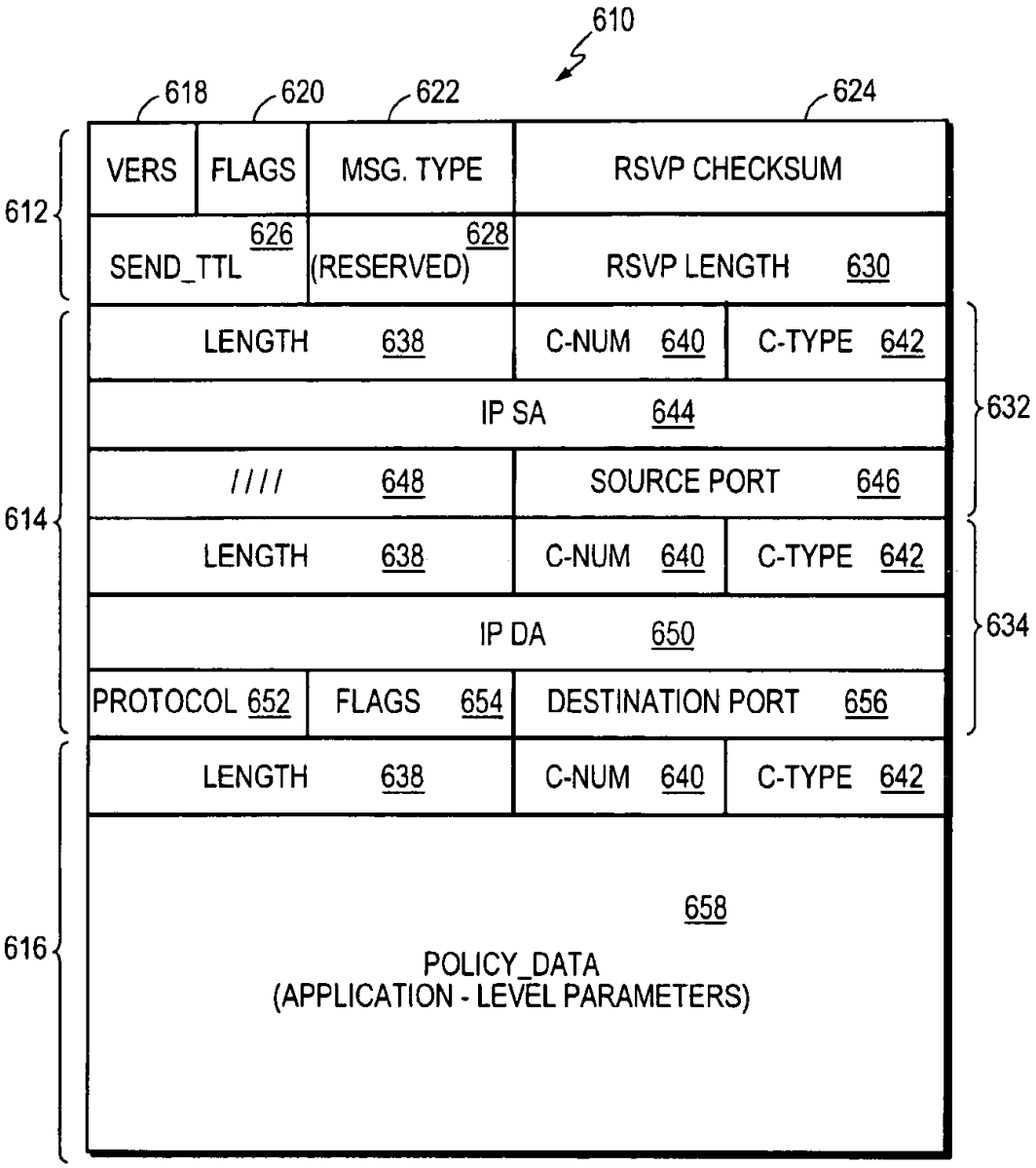

610
618
620
622
624
VERS
FLAGS
MSG. TYPE
RSVP CHECKSUM
612
SEND_TTL 626
(RESERVED) 628
RSVP LENGTH 630
LENGTH 638
C-NUM 640
C-TYPE 642
IP SA 644
632
//// 648
SOURCE PORT 646
614
LENGTH 638
C-NUM 640
C-TYPE 642
IP DA 650
634
PROTOCOL 652
FLAGS 654
DESTINATION PORT 656
LENGTH 638
C-NUM 640
C-TYPE 642
658
616
POLICY_DATA
(APPLICATION - LEVEL PARAMETERS)

FIG. 6

METHOD AND APPARATUS FOR IDENTIFYING NETWORK DATA TRAFFIC FLOWS AND FOR APPLYING QUALITY OF SERVICE TREATMENTS TO THE FLOWS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to the following copending U.S. patent application:

U.S. patent application Ser. No. 09/179,036 entitled, METHOD AND APPARATUS FOR DEFINING AND IMPLEMENTING HIGH-LEVEL QUALITY OF SERVICE POLICIES IN COMPUTER NETWORKS, filed Oct. 26, 1998, now U.S. Pat. No. 6,167,495, and assigned to the assignee of the present application.

FIELD OF THE INVENTION

The present invention relates generally to computer networks, and more specifically, to a method and apparatus for identifying network data traffic flows and for applying quality of service or policy treatments thereto.

BACKGROUND OF THE INVENTION

A computer network typically comprises a plurality of interconnected entities that transmit (i.e., "source") or receive (i.e., "sink") data frames. A common type of computer network is a local area network ("LAN") which typically refers to a privately owned network within a single building or campus. LANs employ a data communication protocol (LAN standard), such as Ethernet, FDDI or Token Ring, that defines the functions performed by the data link and physical layers of a communications architecture (i.e., a protocol stack), such as the Open Systems Interconnection (OSI) Reference Model. In many instances, multiple LANs may be interconnected by point-to-point links, microwave transceivers, satellite hook-ups, etc. to form a wide area network ("WAN"), metropolitan area network ("MAN") or intranet. These LANs and/or WANs, moreover, may be coupled through one or more gateways to the Internet.

Each network entity preferably includes network communication software, which may operate in accordance with the well-known Transport Control Protocol/Internet Protocol (TCP/IP). TCP/IP basically consists of a set of rules defining how entities interact with each other: In particular, TCP/IP defines a series of communication layers, including a transport layer and a network layer. At the transport layer, TCP/IP includes both the User Data Protocol (UDP), which is a connectionless transport protocol, and TCP which is a reliable, connection-oriented transport protocol. When a process at one network entity wishes to communicate with another entity, it formulates one or more messages and passes them to the upper layer of the TCP/IP communication stack. These to messages are passed down through each layer of the stack where they are encapsulated into packets and frames. Each layer also adds information in the form of a header to the messages. The frames are then transmitted over the network links as bits. At the destination entity, the bits are re-assembled and passed up the layers of the destination entity's communication stack. At each layer, the corresponding message headers are also stripped off, thereby recovering the original message which is handed to the receiving process.

One or more intermediate network devices are often used to couple LANs together and allow the corresponding entities to exchange information. For example, a bridge may be used to provide a "bridging" function between two or more LANs. Alternatively, a switch may be utilized to provide a "switching" function for transferring information, such as data frames or packets, among entities of a computer network. Typically, the switch is a computer having a plurality of ports that couple the switch to several LANs and to other switches. The switching function includes receiving data frames at a source port and transferring them to at least one destination port for receipt by another entity. Switches may operate at various levels of the communication stack. For example, a switch may operate at layer 2 which, in the OSI Reference Model, is called the data link layer and includes the Logical Link Control (LLC) and Media Access Control (MAC) sub-layers.

Other intermediate devices, commonly referred to as routers, may operate at higher communication layers, such as layer 3, which in TCP/IP networks corresponds to the Internet Protocol (IP) layer. IP data packets include a corresponding header which contains an IP source address and an IP destination address. Routers or layer 3 switches may re-assemble or convert received data frames from one LAN standard (e.g., Ethernet) to another (e.g. Token Ring). Thus, layer 3 devices are often used to interconnect dissimilar subnetworks. Some layer 3 intermediate network devices may also examine the transport layer headers of received messages to identify the corresponding TCP or UDP port numbers being utilized by the corresponding network entities. Many applications are assigned specific, fixed TCP and/or UDP port numbers in accordance with Request for Comments (RFC) 1700. For example, TCP/UDP port number 80 corresponds to the hyper text transport protocol (HTTP), while port number 21 corresponds to file transfer protocol (ftp) service.

Allocation of Network Resources

Computer networks include numerous services and resources for use in moving traffic throughout the network. For example, different network links, such as Fast Ethernet, Asynchronous Transfer Mode (ATM) channels, network tunnels, satellite links, etc., offer unique speed and bandwidth capabilities. Particular intermediate devices also include specific resources or services, such as number of priority queues, filter settings, availability of different queue selection strategies, congestion control algorithms, etc.

Individual frames or packets, moreover, can be marked so that intermediate devices may treat them in a predetermined manner. For example, the Institute of Electrical and Electronics Engineers (IEEE), in an appendix (802.1p) to the 802.1D bridge standard, describes additional information for the MAC header of Data Link Layer frames. FIG. 1 is a partial block diagram of a Data Link frame 100 which includes a MAC destination address (DA) field 102, a MAC source address (SA) field 104 and a data field 106. In accordance with the 802.1Q standard, a user_priority field 108, among others, is inserted after the MAC SA field 104. The user_priority field 108 may be loaded with a predetermined value (e.g., 0-7) that is associated with a particular treatment, such as background, best effort, excellent effort, etc. Network devices, upon examining the user_priority field 108 of received Data Link frames 100, apply the corresponding treatment to the frames. For example, an intermediate device may have a plurality of transmission priority queues per port, and may assign frames to different queues of a destination port on the basis of the frame's user priority value.

FIG. 1B is a partial block diagram of a Network Layer packet 120 corresponding to the Internet Protocol. Packet

120 includes a type_of_service (ToS) field 122, a protocol field 124, an IP source address (SA) field 126, an IP destination address (DA) field 128 and a data field 130. The ToS field 122 is used to specify a particular service to be applied to the packet 120, such as high reliability, fast delivery, accurate delivery, etc., and comprises a number of sub-fields (not shown). The sub-fields include a three bit IP precedence (IPP) field and three one bit flags (Delay, Throughput and Reliability). By setting the various flags, an entity may indicate which overall service it cares most about (e.g., Throughput versus Reliability). Version 6 of the Internet Protocol (I Pv6) similarly defines a traffic class field, which is also intended to be used for defining the type of service to be applied to the corresponding packet.

Recently, a working group of the Internet Engineering Task Force (IETF), which is an independent standards organization, has proposed replacing the ToS field 112 of Network Layer packets 120 with a one octet differentiated services (DS) field 132 that can be loaded with a differentiated services codepoint. Layer 3 devices that are DS compliant apply a particular per-hop forwarding behavior to data packets based on the contents of their DS fields 132. Examples of per-hop forwarding behaviors include expedited forwarding and assured forwarding. The DS field 132 is typically loaded by DS compliant intermediate devices located at the border of a DS domain, which is a set of DS compliant intermediate devices under common network administration. Thereafter, interior DS compliant devices along the path simply apply the corresponding forwarding behavior to the packet 120.

FIG. 1C is a partial block diagram of a Transport Layer packet 150. The network layer packet 150 preferably includes a source port field 152, a destination port field 154 and a data field 156, among others. Fields 152 and 154 are preferably loaded with the predefined or dynamically agreed-upon TCP or UDP port numbers being utilized by the corresponding network entities.

Service Level Agreements

To interconnect dispersed computer networks, many organizations rely on the infrastructure and facilities of internet service providers (ISPs). For example, an organization may lease a number of T1 lines to interconnect various LANs. These organizations and ISPs typically enter into service level agreements, which include one or more traffic specifiers. These traffic specifiers may place limits on the amount of resources that the subscribing organization will consume for a given charge. For example, a user may agree not to send traffic that exceeds a certain bandwidth (e.g., 1 Mb/s). Traffic entering the service provider's network is monitored (i.e., "policed") to ensure that it complies with the relevant traffic specifiers and is thus "in-profile". Traffic that exceeds a traffic specifier (i.e., traffic that is "out-of-profile") may be dropped or shaped or may cause an accounting change (i.e., causing the user to be charged a higher rate). Another option is to mark the traffic as exceeding the traffic specifier, but nonetheless allow it to proceed through the network. If there is congestion, an intermediate network device may drop such "marked" traffic first in an effort to relieve the congestion.

Multiple Traffic Flows

A process executing at a given network entity, moreover, may generate hundreds if not thousands of traffic flows that are transmitted across the corresponding network every day. A traffic flow generally refers to a set of messages (frames and/or packets) that typically correspond to a particular task, transaction or operation (e.g., a print transaction) and may be identified by 5 network and transport layer parameters (e.g., source and destination IP addresses, source and destination TCP/UDP port numbers and transport protocol). Furthermore, the treatment that should be applied to these different traffic flows varies depending on the particular traffic flow at issue. For example, an on-line trading application may generate stock quote messages, stock transaction messages, transaction status messages, corporate financial information messages, print messages, data back-up messages, etc. A network administrator, moreover, may wish to have very different policies or service treatments applied to these various traffic flows. In particular, the network administrator may want a stock quote message to be given higher priority than a print transaction. Similarly, a $1 million stock transaction message for a premium client should be assigned higher priority than a $100 stock transaction message for a standard customer. Most intermediate network devices, however, lack the ability to distinguish among multiple traffic flows, especially those originating from the same host or server.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method and apparatus for identifying one or more traffic flows from a source entity.

It is a further object of the present invention to provide a method and apparatus for obtaining traffic policies to be applied to identified traffic flows.

It is a further object of the present invention to manage traffic flows in accordance with corresponding policies.

Briefly, the invention relates to a method and apparatus for identifying specific traffic flows originating from a network entity and for applying predetermined policy or service treatments to those flows. In particular, a network entity includes a flow declaration component that is coupled to one or more application programs executing on the entity. The network entity also includes a communication facility that supports message exchange between the application program and other network entities. The flow declaration component includes a message generator and an associated memory for storing one or more traffic flow data structures. For a given traffic flow, the application program calls the flow declaration component and provides it with one or more identifying parameters corresponding to the given flow. In particular, the application program may provide network and transport layer parameters, such as IP source and destination addresses, TCP/UDP port numbers and transport protocol associated with the given traffic flow. It also provides one or more application-level parameters, such as a transaction-type (e.g., a stock transaction), a sub-transaction-type (e.g., a $1 Million stock purchase order), etc. The flow declaration component provides this information to a local policy enforcer, which, in turn, may query a policy server to obtain one or more policy or service treatments that are to be applied to the identified traffic flow. The local policy enforcer then monitors the traffic originating from the network entity and, by examining IP source and destination addresses, among other information, applies the prescribed policy or service treatments to the given traffic flow.

In the preferred embodiment, the application program and the flow declaration component at the network entity interact through an Application Programming Interface (API) layer, which includes a plurality of system calls. In addition, the

flow declaration component generates and transmits one or more application parameter declaration (APD) messages to the local policy enforcer. The APD messages contain the network and transport layer parameters (e.g., IP source and destination addresses, TCP/UDP port numbers and transport protocol) stored at the traffic flow data structure for the given flow. The messages may also contain the application-level parameters specified by the application program. The information, moreover, may be in the form of objects generated by the flow declaration component. Preferably, the flow declaration component and the local policy enforcer exchange messages in accordance with a novel protocol that defines a message scheme in addition to a message format. The local policy enforcer and the policy server may utilize the Common Open Policy Service (COPS) protocol to request and receive particular policies or service treatment rules. Preferably, the policy server maintains or otherwise has access to a store of network policies established by the network administrator.

In another aspect of the invention, the local policy enforcer may establish a traffic flow state that includes the policy or service treatments specified by the policy server. It then monitors the traffic flows originating from the network entity looking for the given traffic flow. Once the given traffic flow is identified, the local policy enforcer applies the policy or service treatments set forth in the corresponding traffic flow state. For example, the policy enforcer may mark the packets or frames with a high priority DS codepoint. When the given traffic flow is complete, the application program may notify the flow declaration component, which, in turn, signals the end of the traffic flow to the local policy enforcer. The policy enforcer may request authorization from the policy server to release or otherwise discard the respective traffic flow state.

In an alternative embodiment of the invention, policy rules may be cached at the local policy enforcer to eliminate the need to query the policy server for each new traffic flow.

In another embodiment of the invention, the APD messages are replaced with one or more enhanced Path or Reservation messages as originally specified in the Resource ReSerVation Protocol (RSVP).

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings, in which:

FIGS. 1A–1C, previously discussed, are partial block diagram of network messages;

FIG. 2 is a highly schematic block diagram of a computer network;

FIG. 3 is a highly schematic, partial block diagram of local policy enforcer;

FIGS. 4A–4D are flow diagrams illustrating the message scheme and tasks performed in identifying a traffic flow and obtaining the corresponding policies;

FIGS. 5A–5B are highly schematic block diagrams illustrating the preferred format of an application parameter declaration message; and

FIG. 6 is a highly schematic block diagram illustrating an enhanced Resource ReSerVation Protocol (RSVP) message in accordance with the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 2 is a highly schematic block diagram of a computer network **200**. The network **200** includes a plurality of local area networks (LANs) **202**, **204** and **206** that are interconnected by a plurality of intermediate network devices **208**, **210**. Coupled to the LANs are a plurality of entities, such as end station **212** and print server **214**. The network further includes at least one policy server **216** that may be coupled to a repository **218** and to a network administrator's station **220**. A server suitable for use as policy server **216** is any Intel x86/Windows NT® or Unix-based platform. The network **200** also includes at least one host or server **222** configured in accordance with the present invention.

In particular, the host/server **222** includes at least one application program or process **224**, a flow declaration component **226** and a communication facility **228**. The flow declaration component **226** includes a message generator **230** that is in communicating relation with the communication facility **228**. Component **226** is also coupled to an associated memory **232** for storing one or more traffic flow data structures **234**. The application program **224** is in communicating relation with both the communication facility **228** and, through an Application Programming Interface (API) layer **236**, to the flow declaration component **226**. The communication facility **228**, in turn, is connected to network **200** via LAN **206**. The host/server **222** also comprises conventional programmable processing elements (not shown), which may contain software program instructions pertaining to the methods of the present invention. Other computer readable media may also be used to store the program instructions.

The communication facility **228** preferably includes one or more software libraries for implementing a communication protocol stack allowing host/server **222** to exchange messages with other network entities, such as end station **212**, print server **214**, etc. In particular, the communication facility **228** may include software layers corresponding to the Transmission Control Protocol/Internet Protocol (TCP/IP), the Internet Packet Exchange (IPX) protocol, the AppleTalk protocol, the DECNet protocol and/or NetBIOS Extended User Interface (NetBEUI). Communication facility **228** further includes transmitting and receiving circuitry and components, including one or more network interface cards (NICs) that establish one or more physical ports to LAN **206** or other LANs for exchanging data packets and frames.

Intermediate network devices **208**, **210** provide basic bridging functions including filtering of data traffic by medium access control (MAC) address, "learning" of a MAC address based upon a source MAC address of a frame and forwarding of the frame based upon a destination MAC address or route information field (RIF). They may also include an Internet Protocol (IP) software layer and provide route processing, path determination and path switching functions. In the illustrated embodiment, the intermediate network devices **208**, **210** are computers having transmitting and receiving circuitry and components, including network interface cards (NICs) establishing physical ports, for exchanging data frames. Intermediate network device **210**, moreover, is preferably configured as a local policy enforcer for traffic flows originating from host/server **222**, as described below.

It should be understood that the network configuration **200** of FIG. **2** is for illustrative purposes only and that the present invention will operate with other, possibly far more complex, network topologies. For example, the repository **218** and network administrator's station **220** may be directly or indirectly connected to the policy server **216** (e.g., through one or more intermediate devices).

FIG. **3** is a partial block diagram of local policy enforcer **210**. Local policy enforcer **210** includes a traffic flow state

machine engine **310** for maintaining flow states corresponding to host/server **222** traffic flows, as described below. The traffic flow state machine engine **310** is coupled to a communication engine **312**. The communication engine **312** is configured to formulate and exchange messages with the policy server **216** and the flow declaration component **226** at host/server **222**. That is, communication engine **312** includes or has access to conventional circuitry for transmitting and receiving messages over the network **200**. The traffic flow state machine engine **310** is also coupled to several traffic management resources and mechanisms. In particular, traffic flow state machine engine **310** is coupled to a packet/frame classifier **314**, a traffic conditioner entity **316**, a queue selector/mapping entity **318** and a scheduler **320**. The traffic conditioner entity **316** includes several sub-components, including one or more metering entities **322**, one or more marker entities **324**, and one or more shaper/dropper entities **326**. The queue selector/mapping entity **318** and scheduler **320** operate on the various queues established by local policy enforcer **210** for its ports and/or interfaces, such as queues **330***a*–**330***e* corresponding to an interface **332**.

The term intermediate network device is intended broadly to cover any intermediate device for interconnecting end stations of a computer network, including, without limitation, layer 3 devices or routers, as defined by Request for Comments (RFC) **1812** from the Internet Engineering Task Force (IETF), intermediate devices that are only partially compliant with RFC **1812**, intermediate devices that provide additional functionality, such as Virtual Local Area Network (VLAN) support, IEEE 802.1Q support and/or IEEE 802.1D support, etc. Intermediate network device also includes layer 2 intermediate devices, such as switches and bridges, including, without limitation, devices that are fully or partially compliant with the IEEE 802.1D standard and intermediate devices that provide additional functionality, such as VLAN support, IEEE 802.1Q support and/or IEEE 802.1p support, Asynchronous Transfer Mode (ATM) switches, Frame Relay switches, etc.

FIGS. 4A–4D are flow diagrams illustrating a preferred message scheme, relative to time t, in accordance with the present invention. In general, application program **224** identifies one or more anticipated traffic flows to the flow declaration component **226**, which, in turn, notifies the local policy enforcer **210**. The local policy enforcer **210** requests and receives from the policy server **216** corresponding policy or service treatments for the anticipated traffic flows. Local policy enforcer **210** then monitors the traffic originating from host/server **222** to identify those frames and/or packets corresponding to the identified flows. When such a flow is detected, local policy enforcer **210** applies the specified policy or service treatments to corresponding data frames and/or packets.

Identification of Traffic Flows

Assume that application program **224** is a stock transaction program that can provide stock quotes to and process stock transactions from remote clients, such as end station **212**. The application program **224** preferably communicates with end station **212** across network **200** through the communication facility **228** at host/server **222** in a conventional manner. Program **224** also communicates with the flow declaration component **226** preferably through a plurality of application programming interface (API) system calls to API layer **236**. These API calls are generally issued by the program **224** along with one or more arguments and may be returned by the flow declaration component **226**.

In particular, upon initialization at host/server **222**, the application program **224** preferably issues a StartUp( ) API call **410** to the API layer **236** at flow declaration component **226**. Program **226** preferably loads the StartUp( ) call **410** with an application identifier that uniquely identifies application program **224** to component **226** as an argument. The application identifier may be a globally unique identifier (GUID), which is a 128 bit long value typically provided by the application developer, although other identifiers may also be used (e.g., application name). The StartUp( ) call **410** may be returned by the flow declaration component **226** with a version number as an argument. The version number corresponds to the version of software being executed by the flow declaration component **226**. Other arguments, such as the quality-of-service (QoS) and/or traffic management resources that are available to traffic flows originating from program **224**, may also be returned by flow declaration component **226**.

For example, assume end station **212** contacts program **224** and requests a stock quote for a particular equity (e.g., IBM common stock). Program **224** retrieves the requested information and prepares a message containing the requested stock quote for transmission to end station **212**. Before program **224** commences the traffic flow corresponding to requested stock quote, it preferably issues a NewBindings( ) call **412** to the API layer **236** of the flow declaration component **226**. The NewBindings( ) call **412** is used to inform flow declaration component **226** of an anticipated traffic flow to which some policy or service treatments should be applied. In response to the NewBindings( ) call **412**, flow declaration component **226** generates a bindings handle, e.g., H1, and creates a traffic flow data structure **234** within associated memory **232**. Component **226** also maps or associates the traffic flow data structure **234** with the returned bindings handle H1. Flow declaration component **226** also returns the NewBindings( ) call **412** to program **224** with the handle H1 as an argument.

Next, traffic flow data structure **234** is loaded with information identifying the anticipated traffic flow. More specifically, program **224** next issues one or more network and transport layer parameter "Set" API calls **414**. These Set calls **414** are used by the flow declaration component **226** to load traffic flow data structure **234** with network and transport layer parameters, such as Internet Protocol (IP) addresses and TCP/UDP port is numbers. For example, program **224** may issue a SetSourcePort( ) call **414***a* using the returned handle, H1, and the transport layer port number (e.g., TCP port number **1098**) to be utilized by program **226** as its arguments. In response, flow declaration component **226** loads the identified source port number (i.e., **1098**) into the traffic flow data structure **234** corresponding to handle H1. Flow declaration component **226** may return an acknowledgment to program **224** as an argument to the SetSourcePort( ) call **414***a*. If a problem arises, flow declaration component **226** may return an error message (e.g., insufficient memory, unknown handle, out of bound port number, etc.) as the argument.

In a similar manner, program **224** preferably causes the flow declaration component **226** to load the corresponding traffic flow data structure **234** with its IP address, the transport layer protocol (e.g., TCP) and the destination port number and IP address of the receiving process at end station **212**. More specifically, in addition to the SetSourcePort( ) call **414***a*, program **224** may issue one or more of the following API system calls:

SetSourceIP( ) **414***b*;
SetTransportProtocol( ) **414***c*;
SetDestinationPort( ) **414***d*; *and*
SetDestinationIP( ) **414***e*.

Again, program **224** uses the previously returned handle, H1, and the corresponding information (e.g., IP address, transport protocol or port number) as arguments to these API calls. As each Set API call **414** is received, the flow declaration component **226** loads the identified parameter into the traffic flow data structure **234**. Flow declaration component **226** may similarly return the Set API call **414** with an error code or an acknowledgment as an argument. It should be understood that additional "Set" API calls **414** may be defined depending on the format of the included information. For example, by utilizing a SetSourceIPByLong( ) call (not shown), program **224** may specify its IP address as a 32 bit binary sequence. Alternatively, by utilizing a SetSourceIPByString( ) call (not shown), program **224** may specify its IP address in dotted decimal format (e.g., **128.120.52.123**) or as a host name (e.g., name.department.company.domain). In addition, a single SetNetworkTransportParameters( ) system call may be defined to set all of the network and transport layer parameters at once.

It should be understood that application program **224** may obtain IP source and destination addresses, port numbers and transport protocol for use in communicating with end station **212** from the communication facility **228** in a conventional manner. It should be further understood that application program **224** may utilize one or more wildcards when specifying the network and transport layer parameters.

In addition to the network and transport layer parameters (e.g., source and destination IP addresses, transport protocol and source and destination TCP/UDP port numbers) which correspond to a particular flow of traffic, program **236** may specify other identifying characteristics and/or policy elements of the anticipated traffic flow. That is, program **224** may issue one or more application-level "Set" API calls **416** to the flow declaration component **226**. For example, a SetInteger( ) call **416***a* may be used to specify some numerical aspect (e.g., the size of a file being transferred) of the anticipated traffic flow. The arguments of the SetInteger( ) call **416***a* include the handle H1, the numeric policy element (e.g., 786 Kbytes) and a policy element identifier (PID) that maps the numeric policy element to a particular type or class of information (e.g., file size). When the traffic type data structure **234** is subsequently transferred to and processed by other entities, as described below, the PID will identify its corresponding information. In response to the SetInteger( ) call **416***a*, flow declaration component **226** loads the traffic flow data structure **234** with the numeric policy element and the PID. Flow declaration component **226** may return the SetInteger( ) call **416***a* to program **224** with an acknowledgment or error message as arguments.

Other application-level Set calls may also be defined. For example, a SetFloat( ) call **416***b* is used to associate a numeric value represented in floating decimal format with the anticipated traffic flow. A SetString( ) call **416***c* may be used to associate an alpha-numeric string with the anticipated flow. For example, if the anticipated traffic flow is to contain a video segment, program **224** may identify the name of the particular video segment and/or the viewer by utilizing the SetString( ) call **416***c*. Program **224** uses the handle H1 and the particular alpha-numeric string as arguments for the SetString( ) call **416***c*. A PID that maps an alpha-numeric string to name of a video segment is also included. This information is similarly loaded into the corresponding traffic flow data structure **234** by the flow declaration component **226**. A generic Set( ) call **416***d* may be used for specifying traffic flow characteristics that do not correspond to integer, floating decimal point or alpha-numeric string formats. For example, program **224** may specify a policy element in the well-known eXternal Data Representation (XDR) format. This XDR policy element is included as an argument in the Set( ) call **416***d* to the flow declaration component **226**, which, in response, simply copies the XDR policy element into traffic flow data structure **234**. The policy element may alternatively be specified using the well-known Abstract Syntax Notation One (ASN.1) format, or any other similar translation or encoding techniques.

The application-level parameters may encompass a whole range of information relating to different aspects of the traffic flow from the application program **224**. For example, application-level parameters include such information as user name (e.g., John Smith), user department (e.g., engineering, accounting, marketing, etc.), application name (e.g., SAP R/3, PeopleSoft, etc.), application module (e.g., SAP R/3 accounting form, SAP R/3 order entry form, etc.), transaction type (e.g., print), sub-transaction type (e.g., print on HP Laser Jet Printer), transaction name (e.g., print monthly sales report), sub-transaction name (e.g., print monthly sales report on A4 paper), application state (e.g., normal mode, critical mode, primary mode, back-up mode, etc.). For a video streaming application, the application-level parameters might include user name, film name, film compression method, film priority, optimal bandwidth, etc. Similarly, for a voice over IP application, the application-level parameters may include calling party, called party, compression method, service level of calling party (e.g., gold, silver, bronze), etc. In addition, for World Wide Web (WWW) server-type applications, the application-level parameters may include Uniform Resource Locator (URL) (e.g., http://www.altavista.com/cgi-in/query?pg=aq&kl=en&r=&search=Search&q=Speech+ne ar+recognition), front-end URL (e.g., http://www.altavista.com), back-end URL (e.g., query?pg=aq&kl=en&r=&search=Search&q=Speech+near+recognition), mime type (e.g., text file, image file, language, etc.), file size, etc. Those skilled in the art will recognize that many other application-level parameters may be defined.

Application program **224** can also retrieve information stored at the traffic flow data structure **234** by issuing one or more Get API system calls **418** (FIG. 4B). For example, program **224** may issue a GetSourcePort( ) call **418***a* using the returned bindings handle HI as an argument. In response, flow declaration component **226** parses the traffic flow data structure **234** and retrieves the source port information stored therein. Component **226** then returns the GetSourcePort( ) call **418***a* to program **224** with the source port as an argument. Program **224** may issue similar Get API calls to retrieve other network and transport layer parameters stored at the traffic flow data structure **234**.

It should be understood that additional "Get" API system calls may be defined for retrieving application-level information from the traffic flow data structure **234**.

After issuing the application-level Set API calls **416**, if any, the corresponding traffic flow data structure **234** is complete. That is, data structure **234** has been loaded with each of the identifying characteristics specified by the application program **224** for the anticipated traffic flow.

In accordance with the invention, the flow declaration component **226** also opens a communication session with the local policy enforcer **210** and exchanges one or more Application Parameters Declaration (APD) messages. In the preferred embodiment, the flow declaration component **226** opens a reliable, connection-based "socket" session using

the well-know Transport Control Protocol (TCP) protocol of the TCP/IP communication protocol stack. A "socket" is essentially an interface between the application and transport layers of a communication protocol stack that enables the transport layer to identify which process it must communicate with in the application layer. A socket interfaces to a TCP/IP communication protocol stack via APIs consisting of a set of entry points into the stack. Applications that require TCP/IP connectivity thus use the socket APIs to interface into the TCP/IP stack. For a connection-oriented protocol (such a TCP), the socket may be considered a "session".

It should be understood that other protocols, including but not limited to connectionless protocols such as UDP, may be used to establish communication between the flow declaration component **226** and the local policy enforcer **210**. Additionally, component **226** may communicate with local policy enforcer **210** at the network layer by addressing IP format APD messages to end station **212** (i.e., using the same destination address as the anticipated traffic flow) with the well-known Router Alert IP option asserted. Here, local policy enforcer **210** will intercept such asserted network layer packets and may act on them itself and/or forward them to some other network device.

Component **226** may be preconfigured with the IP address of the local policy enforcer **210** or it may dynamically obtain the address of a local policy enforcer. For example, component **226** or application program **224** may broadcast an advertisement seeking the IP address of an intermediate network device that is capable of obtaining and applying policy or service treatments to the anticipated traffic flow from program **224**. Local policy enforcer **210** is preferably configured to respond to such advertisements with its IP address.

Component **226** may receive a "virtual" address that corresponds to a group of available local policy enforcers in a manner similar to the Standby Router Protocol described in U.S. Pat. No. 5,473,599, which is hereby incorporated by reference in its entirety. A single "active" local policy enforcer may be elected from the group to perform the functions described herein.

It should be further understood that the flow declaration component **226** preferably opens one TCP session with the local policy enforcer **210** per application program **224** per network interface card (NIC). More specifically, if host/server **222** is connected to network **200** through multiple LANs (each with a corresponding NIC), then traffic flows from program **224** may be forwarded onto any of these LANs. To ensure that the appropriate policy or service treatments are applied regardless of which LAN initially carries the flow, flow declaration component **226** preferably establishes a separate communication session with a local policy enforcer **210** through each LAN (i.e., through each NIC) for every program **224** that requests services from component **226**.

In particular, flow declaration component **226** directs message generator **230** to formulate a Client Open message **420** for forwarding to the local policy enforcer **210**. The Client Open message **420** establishes communication between the local policy enforcer **210** and the flow declaration component **226** and may be used to determine whether the local policy enforcer **210** has the resources to monitor the anticipated flow from the application program **224** and to apply the appropriate policy or service treatments. FIG. **5A** is a block diagram of the preferred format of the Client Open message **420**. In particular, the Client Open message **420** includes at least two elements: a header **510** and a timer area **512**. The header **510** includes a version field **516**, a flags field **518**, an operation code field **520** and a message length field **524**. It may also include one or more unused fields, such as field **522**. Version field **516** preferably contains the version of the software being implemented at the flow declaration component **226**. Flags field **518** preferably contains at least one flag that may be asserted or de-asserted by the flow declaration component **226**, as described below. The operation code field **520** indicates the type of APD message. For a Client Open message **420**, for example, field **520** is preferably loaded with the value "7". The message length field **524** specifies the length (in octets) of the Client Open message **420**.

The timer area **512** includes a length field **526** which specifies the length (preferably in octets) of the timer area **512**, a Class Number (C-Num) field **528**, a Class Type (C-Type) field **530** and a Keep Alive Timer Value field **532**. Timer area **512** may also include one or more unused fields, **534**, **536**. The Class Number field **528** is loaded with an agreed-upon value (e.g., "11") indicating that this portion of the Client Open message **420** (i.e., timer area **512**) contains a keep alive timer value. Where multiple types may exist for a given class number, the Class Type field **530** is used to specify the particular type. Here, field **530** is preferably set to "1". Flow declaration component **226** preferably loads the Keep Alive Timer Value field **532** with a proposed time value (e.g., 30 seconds) to be used for maintaining the TCP session in the absence of substantive APD messages, as described below.

Message generator **230** preferably passes the Client Open message **420** down to the communication facility **228** where it is encapsulated into one or more TCP packets and forwarded to the local policy enforcer **210** in a conventional manner. The APD messages, such as the Client Open message **420**, preferably use a well-known destination port number, such as **1022**. The source destination port for the flow declaration component **226** may be dynamically agreed-upon when the TCP session with the local policy enforcer **210** is first established. At the local policy enforcer **210**, message **420** is received at the communication engine **312** and passed up to the traffic flow state machine engine **310**. The traffic flow state machine engine **310** examines the message **420** which it recognizes as a Client Open message due to the value (e.g., "7") loaded in the operation code field **520**. Local policy enforcer **210** may first determine whether it has adequate resources to accept a new client. For example, local policy enforcer **210** may include an admission control module (not shown) that determines the percentage of time that its central processing unit (CPU) has remained idle recently, its available memory (for storing policies associated with component **226**) and the availability of its traffic management resources, such as meter **322**, marker **324** and shaper/dropper **326**, to manage additional traffic flows.

Assuming local policy enforcer **210** has sufficient available resources, it replies to the flow declaration component **226** with a Client Accept message **422**. The format of the Client Accept message **422** is similar to the format of the Client Open message **422** shown in FIG. **5A**. In particular, the Client Accept message **422** also includes a header that is similar to header **510** and a timer area that is similar to timer area **512**. The operation code for the Client Accept message **422** (which is loaded in field **520**) is another predefined value (e.g., "8") so that flow declaration component **226** will recognize this APD message as a Client Accept message. The traffic flow state machine engine **310** also loads a value in the Keep Alive Timer Value field **532** which may corre-

spond to the value proposed by component **226** or may be a new value selected by the local policy enforcer **210**.

The traffic flow state machine engine **310** hands the Client Accept message **422** to its communication engine **312** which may encapsulate the message as required and forwards it to the host/server **222**. At the host/server **222** the message is received at the communication facility **228** and passed up to the flow declaration component **226** where it is examined. Flow declaration component **226** examines the operation code field **520** and "learns" that it is a Client Accept message. Flow declaration component **226** also examines the keep alive timer field **532** to determine what value has been specified by local policy enforcer **210**, which is used to generate additional APD messages, as described below.

It should be understood that the flow declaration component **226** may issue the Client Open message **420** as soon as the StartUp( ) call **420** is issued if not earlier.

When application program **224** is ready to begin transmitting the anticipated traffic flow (e.g., the IBM stock quote form) to end station **212**, it issues a BeginFlow( ) call **424***a* to the flow declaration component. Preferably, the BeginFlow( ) call **424***a* is issued slightly before (e.g., 50 ms) program **224** begins forwarding the message to the communication facility **228**. It should be understood, however, that the BeginFlow( ) call **424***a* may be issued at the same time as the anticipated flow to end station **212** is commenced or even slightly later. The application program **224** uses the previously returned handle H1 as an argument to the BeginFlow( ) call **424***a*. If program **224** wishes to receive any feedback regarding the policy or service treatments that are applied to the respective traffic flow, it may also assert a flag argument in the BeginFlow( ) call **424***a* and add one or more callback functions as additional arguments. The callback function preferably identifies an entry point in the application program **224** to which the requested feedback is to be returned. Program **224** may also load other information or data that will simply be returned to it with the requested feedback to assist program **224**, for example, in mapping the returned feedback to a particular task.

The BeginFlow( ) call **424** is received and examined by the flow declaration component **226**, which, in part, determines whether the feedback flag has been set. If so, it also looks for any callback functions and information arguments specified by program **224**. Flow declaration component **226** may also return a flow handle, H2, to program **224** as an argument to the BeginFlow( ) call **424**. Component **226** may also return an acknowledgment or error message as additional arguments. Assuming that the BeginFlow( ) call **424** did not cause any errors, flow declaration component **226** then directs its message generator **230** to formulate a Flow Start APD message **426**.

FIG. **5B** is a block diagram of a preferred Flow Start message **426**, which is similar to the Client Open message **420**. In particular, the Flow Start message **426** includes a header **510** having a flags field **518** and an operation code field **520**, among others. If program **224** requested policy feedback, then message generator **230** preferably asserts the flag in field **518**. In addition, the operation code field **520** is preferably loaded with the value "1" to indicated that this particular APD message is a Flow Start message **426**. Following the header **510** is a handle area **540**, which includes a length field **542** (specifying the length of the handle area **540**), a Class Number (C-Num) field **544**, a Class Type (C-Type) field **546**, a device handle field **548** and a flow handle field **550**. The C-Num field **544** is loaded with an agreed-upon value (e.g., "1") indicating that this portion of the Flow Start message **426** contains a flow handle. The C-Type field **546** may also be set to "1". The device handle field **548** preferably contains a 2 octet identifier selected by the local policy enforcer **210** during establishment of the communication session. For example, the device handle may be "**1327**". The flow handle field **550** preferably contains the flow handle H2 generated by the flow declaration component **226** in response to the BeginFlow( ) call **424**.

Following the handle area **540** are a plurality of policy bindings **552**, such as policy bindings **552***a*, **552***b* and **552***c*. The policy bindings **552** contain encoded versions of the information stored in the traffic flow data structure **234** that corresponds to the flow handle specified in field **550**. Each policy binding **552**, moreover, has two elements, a policy identifier element **554** and an encoded policy instance element **556**. Basically, the policy identifier element **554** identifies the type or instance of policy element that is contained in the associated encoded policy instance element **556**. Each policy identifier element **554** includes a plurality of fields, including a length field **558** (specifying its length), a policy identifier (Policy ID) type field **560** and a policy identifier field **562**. Each encoded policy instance element **556** similarly includes a plurality of fields, including a length field **564** (specifying its length), an encapsulation type field **566** and an encoded policy element field **568**.

The first policy binding **552***a*, for example, may contain an encoded copy of the source port identified by program **224** with the SetSourcePort( ) call **414***a* and stored at the respective traffic flow data structure **234**. More specifically, message generator **230** loads policy identifier field **562***a* with the type or instance of the policy element (e.g., "source port"). In the preferred embodiment, this name is a Policy Identifier (PID) as specified in the Internet Engineering Task Force (IETF) draft document *COPS Usage for Differentiated Services* submitted by the Network Working Group, dated December 1998, and incorporated herein by reference in its entirety. A PID specifies a particular policy class (e.g., a type of policy data item) or policy instance (e.g., a particular instance of a given policy class) in a hierarchical arrangement. The Policy ID type field **560***a* contains a predefined value reflecting that field **562***a* contains information in PID format. Component **226** preferably includes a Policy Information Base (PIB) for use in deriving the particular policy identifiers, as described in *COPS Usage for Differentiated Services*.

The message generator **230** then accesses the source port information from the respective traffic flow data structure **234** and translates it into a machine independent format suitable for transmission across network **200**. For example, the source port information may be translated in accordance with the ASN.1 translation technique. The encapsulated version of the source port is then loaded in the encoded policy element field **568***a* of binding **552***a*. The encapsulation type field **566***a* contains a predefined value reflecting that the information in field **568***a* has been encapsulated according to ASN.1. Message generator **230** similarly builds additional bindings **552** that contain encapsulated versions of the source IP address, transport protocol, destination port number and destination IP address as specified by program **224** in API calls **414***b*–**414***e* and stored at traffic flow data structure **234**. Message generator **230** also formulates separate bindings **552** for each of the application-level data items established by the application program **224** through application-level API calls **416**. Again, each of these application-level data items may be identified by a corresponding PID which is loaded in the Policy ID type field **562** of the respective binding **552**. The application-level data item is then translated into a machine-independent format

(e.g., through ASN.1) and loaded in the respective encoded policy element field **568**, as described above.

It should be understood that other translation techniques, such as XDR, may also be used. It should be further understood that the contents of other fields, including policy identifier field **556**, should be similarly translated into machine-independent format.

The Flow Start message **426** is then handed down to the communication facility **228** for transmission to the local policy enforcer **210**. At the local policy enforcer **210**, the message **426** is captured by the communication engine **312** and handed to the traffic flow state machine engine **310** which parses the operation code field **520** to determine that the message is a Flow Start APD message. In response, the local policy enforcer **210** proceeds to obtain the particular policy rules or service treatments that are to be applied to this flow (e.g., a stock quote form for IBM). In particular, the local policy enforcer **210** formulates a Request Policy message **428** for transmission to the policy server **216**. In the preferred embodiment, the format of the Request Policy message **428** corresponds to the Request message of the Common Open Policy Service (COPS) Protocol specified in the IETF draft document *The Common Open Policy Service (COPS) Protocol*, dated Aug. 6, 1998, and incorporated herein by reference in its entirety.

According to the COPS protocol, Request messages include a plurality of flags, such as a request type flag and a message flag, and a plurality of objects. The request type flag for message **428** is preferably set to the COPS value that corresponds to "Incoming-Message/Admission Control Request" type COPS messages and the message type flag should be set to "1". Furthermore, the "In-Interface" object of the Request Policy message **428** is preferably set to the VLAN designation associated with the local policy enforcer's interface at which the Flow Start message **426** was received. The bindings **552** of the Flow Start message **426**, which may not be meaningful to the local policy enforcer **210**, are preferably loaded (i.e., copied as opaque objects) into the Client Specific Information (ClientSI) object portion of the Request Policy message **428**. The local policy enforcer **210** also loads a unique handle that identifies the anticipated traffic flow from program **224** into the Request Policy message **428**. This handle, moreover, is used in all messages exchanged between the local policy enforcer **210** and the policy server **216** for this anticipated traffic flow. The handle may be the flow handle H2 previously returned by the flow declaration component **226**.

It should be understood that intermediate network devices, such as local policy enforcer **210**, may learn of the identity of the policy server **216** through any conventional means, such as manual configuration or a device configuration protocol.

The Request Policy message **428** is received at the policy server **216**, which examines the network parameters specified for the anticipated traffic flow, including the IP addresses, port numbers and transport protocol. The policy server **216** also examines the application-level parameters specified by program **224** and provided to the policy server **216** in the Request Policy message **428**. Based on this information, the policy server **216** makes a decision regarding the policy rules or service treatments to be applied to this traffic flow. For example, as described in co-pending U.S. Patent Application Ser. No. 09/179,036, which is hereby incorporated by reference in its entirety, the policy server **216** may obtain information from the repository **218** and/or network administrator via end station **220** and, in response, formulate one or more traffic management rules, such as classification, behavioral or configuration rules. More specifically, server **216** may formulate one or more classification rules for instructing the local policy enforcer **210** to classify data packets and frames from this traffic flow with a given DS codepoint, IP Precedence and/or user priority. Policy server **216** may also formulate one or more behavioral rules that instruct the local policy enforcer **210** to map packets with the given DS codepoint to a particular queue (e.g., **330***d*) and to apply a particular scheduling algorithm (e.g., WFQ). These policy decisions or rules are then loaded into a Policy Decision message **430** and sent from the policy server **216** to the local policy enforcer **210**.

Communication engine **312** captures the Policy Decision message **430** and forwards it to the traffic flow state machine engine **310**, which, in turn, extracts the policy decisions or rules contained in the message **430**. Traffic flow state machine engine **310** preferably establishes a flow state (not shown) for the anticipated traffic flow that includes information identifying the anticipated traffic flow (such as IP addresses, port numbers and transport protocol) and the policy decisions or rules to be applied to that traffic. Traffic flow state machine engine **310** may also build one or more data structures (such as tables) to store the mappings contained in the Policy Decision message **430**.

As packets or frames are received at the local policy enforcer **210**, they are examined by the packet/frame classifier **314**. More specifically, the packet/frame classifier **314** parses the source and destination port fields **152, 154** (FIG. **1C**) and the IP source and destination address fields **126, 128** and the protocol field **124** (FIG. **1B**). This information is then supplied to the traffic flow state machine engine **310**, which determines whether a traffic flow state has been established for such packets or frames. Assuming the packets or frames correspond to the anticipated flow from the program **224** to end station **212** (e.g., the IBM stock quote form), a traffic flow state will exist and have associated policy rules or service treatments as specified in the Policy Decision message **430** from policy server **216**. Local policy enforcer **210** then applies the specified treatments to these packets or frames. For example, the traffic flow state machine engine **310** may instruct the packet/frame classifier, to set the DS field **132** (FIG. **1B**) of such packets or frames to a value associated with best effort traffic. Similarly, the traffic flow state machine engine **310** may instruct the queue selector/mapping entity **318** to place these packets or frames in a particular (e.g., moderate priority) queue. Alternatively or in addition, packet/frame classifier may be instructed to load the ToS field **122** (FIG. **1B**) or the user_priority field **108** (FIG. **1A**) with predetermined values so as to implement these treatments at other intermediate network devices, such as device **208**.

To the extent the application program **224** requested feedback as to the policy or service treatments applied to this traffic flow, the local policy enforcer **210** may formulate and send one or more Decision Feedback APD messages **432** to the flow declaration component **226**. The Decision Feedback message **432** is similar in format to the Flow Start message **426**. In particular, the Decision Feedback message **432** has a header **510** and a handle area **540**. For Decision Feedback messages **432**, the operation code field **520** is preferably loaded with the value "3". Appended to the handle area **540** are one or more decision bindings (not shown) that are similar in format to the policy bindings **552**. In particular, each decision binding contains a treatment specified by the policy server **216** and applied by the local policy enforcer **210**. For example, a first decision binding may provide that the specified traffic flow is being marked with a particular

DS codepoint. Other decision bindings may specify the IP Precedence or user_priority values being entered in fields **122, 108**, respectively, of this traffic flow. Other decision bindings may be more abstract and describe abstract service classes granted to the traffic flow. The Decision Request message **432** is received at the communication facility **228** and passed up to the flow declaration component **226**. The flow declaration component **228** extracts the particular treatments from the decision bindings and returns them to the application program **224** through a callback function **434** specified by the application program **224** in the BeginFlow( ) call **424**.

In order to maintain the TCP session established between the flow declaration component **226** and the local policy enforcer **210**, the flow declaration component **226** may send one or more Keep Alive APD messages **436**. The Keep Alive message **436** simply includes a header **510** with the operation code field set to "9" and the message length field **524** set to "0". Flow declaration component **226** preferably sends at least one Keep Alive message **436** within every time period specified in the keep alive timer value field **532** of the Client Accept message **422**.

It should be understood that the policy server **216** may unilaterally send a Decision Change message **438** to the local policy enforcer **210** if a change in the previously supplied policy rules or service treatments occurs after the Policy Decision message **430** was sent. For example, the policy server **216** may obtain up-dated information from the repository **218** or from the network administrator through end station **220**. This up-dated information may affect the policy rules or service treatments previously supplied to the local policy enforcer **210**. In response, the policy server **216** preferably formulates and sends the Decision Change message **438**. The format of the Decisions Change message **438** is preferably the same as the Policy Decision message **430**. The Decision Change message **438** is similarly captured at the communication engine **312** of the local policy enforcer **210** and forwarded to the traffic flow state machine engine **310**.

To the extent the Decision Change message **438** includes new policy rules or service treatments, the traffic flow state machine **310** preferably up-dates its traffic flow state accordingly. In addition, the traffic flow state machine **310** applies the up-dated policy rules or service treatments to subsequently received packets or frames that correspond to the traffic flow. The local policy enforcer **210** may also generate and send a Decision Feedback message (like message **432**) to component **226** if feedback was requested by program **224**.

The policy server **216** may also transmit one or more Decision messages to other intermediate network devices, such as device **208**, that are along the path of the anticipated traffic flow from host/server **222** to end station **212**. These Decision messages similarly inform the intermediate network devices as to what policy rules or service treatments to apply to the traffic flow from program **224**, which presumably has already been classified by the local policy enforcer **210**. Policy server **216** is thus able to provide end-to-end quality of service support.

It should be understood that the local policy enforcer **210** and the policy server **216** may exchange additional COPS messages as required, such as COPS Client Open and COPS Client Accept messages among others.

The local policy enforcer **210** may also send one or more Keep Alive APD messages **440** to the flow declaration component **226** at the host/server **222**. The Keep Alive message **440** from the local policy enforcer **210** preferably has the same format as Keep Alive message **436** from component **226**.

It should be further understood that the application program **224** may change certain characteristics associated with the traffic flow if the nature of the flow changes over time. For example, after reviewing the quote for IBM stock, the user at end station **212** may decide to place a "buy" order for IBM stock. In response, application program **224** may transmit a stock transaction form. Furthermore, the policies or service treatments to be applied to the traffic flow corresponding to the stock quote form may be very different from the treatments that should be applied to the traffic flow corresponding to the stock transaction form. Accordingly, the program **224** may issue one or more new application-level Set API calls **442**. For example, the program may issue a SetInteger( ) call **442***a*, a SetString( ) call **442***b*, a SetFloat( ) call **442***c* and/or a Set( ) call **442***d*. These calls are generally the same as the previously described application-level Set API calls **416** and, although the program **224** utilizes the previously returned handle **H1** as an argument, it enters new or updated information (e.g., stock transaction versus stock quote forms). In response, the flow declaration component **226** overwrites the corresponding entries in the respective traffic flow data structure **234** with the new or up-dated information.

The application program **224** then issues a BeginUpdatedFlow( ) call **444** at or about the time that it begins forwarding the stock transaction form to the user at end station **212**. The BeginUpdatedFlow( ) call **444** is preferably the same as the BeginFlow call **424** described above. In response, the flow declaration component **226** directs the message generator **230** to generate and send a Flow Update APD message **446** to the local policy enforcer **210**. The Flow Update message **446** is similar to the Flow Start message **424** and also includes one or more bindings generated from the information stored in the respective traffic flow data structure **234**. Since the information contained in the traffic flow data structure **234** has been up-dated (through the issuance of the Set API calls **442**), the bindings will be different from the bindings appended to the original Flow Start message **426**.

At the local policy enforcer **210**, the Flow Update message **446** is examined and a Request Policy Update message **428** is preferably formulated and sent to the policy server **216**. The Request Policy Update message **428** has the same general format as the original COPS Request Policy message **448**, although it includes the new bindings generated as a result of the Set API calls **442**. The policy server **216** examines the Request Policy Update message **448** and, in response, obtains the appropriate policy rules or service treatments for this up-dated traffic flow. The policy server **216** then loads these up-dated policy rules or service treatments in a Policy Decision Update message **450**, which is sent to the local policy enforcer **210**. Since at least some of the traffic characteristics have changed, the policies or treatments contained in the Policy Decision Update message **450** may be different than the treatments previously provided in the Policy Decision **430**. For example, the up-dated policies may provide that this traffic flow is to be classified as high priority and granted excellent effort treatment. Similarly, the up-dated policies may provide that the DS field **132** of packets or frames from this traffic flow should be loaded with a DS codepoint associated with expedited forwarding.

The Policy Decision Update message **450** is received at the local policy enforcer **210** which modifies the corresponding traffic flow state with the up-dated policies. The local

policy enforcer **210** also applies these up-dated policies to any subsequently received packets or frames from the host/server **222** that satisfy the previously identified network and transport layer parameters (e.g., IP addresses, port numbers and transport protocol). Local policy enforcer **210** may also provide feedback to component **226** as described above.

When the traffic flow between the application program **224** and end station **212** is finished, program **224** preferably issues a ReleaseFlow( ) call **452** to the flow declaration component **226** using the previously returned flow handle H2 as an argument. Flow declaration component **226** may return an acknowledgment or an error message to the program **224**. In response, the flow declaration component **226** directs message generator **230** to formulate a Flow End APD message **454**. The format of the Flow End message **454** is preferably the same as the Flow Start message **426**, although the operation code field **520** is preferably loaded with "2" to signify that it is a Flow End message. Although the flow declaration component **226** forwards the Flow End message **454** to the local policy enforcer **210**, it preferably does not discard the traffic flow data structure **234**.

In response, the local policy enforcer **210** formulates a COPS Request message **546** to inform the policy server **216** that the respective traffic flow is finished. The policy server **216** may reply with a Decision message **458** authorizing the local policy enforcer **210** to erase the traffic flow state which was established for this particular flow. If the application program **224** subsequently initiates another traffic flow with the same end station **212**, it may re-use the information stored in the traffic flow data structure **234** by issuing another BeginFlow( ) call **424** utilizing the previously returned bindings handle H1. The flow declaration component **226**, in response, proceeds as described above by sending a Flow Start message **426** to the local policy enforcer **210**.

The application program **224** may also issue a DestroyBindings( ) call **460** to the flow declaration component **226** whenever it concludes that the bindings are no longer needed. Program **224** preferably utilizes the previously returned bindings handle H1 as an argument to the DestroyBindings( ) call **460**. In response, component **226** preferably discards the contents of the traffic flow data structure **234** that corresponds to bindings handle H1.

When the application program **224** is closed it should shutdown all outstanding traffic flow services by issuing corresponding ReleaseFlow( ) calls **452** and it should also destroy all bindings that it created by issuing DestroyBindings( ) calls **460**. In response, component **226** directs message generator **230** to formulate a Client Close APD message **462**. The Client Close message **462** is simply a header **510** with the operation code field **520** loaded with the value "10". In response, the local policy enforcer **210** formulates and sends a COPS Request message **464** to the policy server **216** indicating that the program **224** is closed. The policy server **216** may reply with a COPS Decision message **466** instructing the local policy enforcer **210** to release all of the corresponding traffic flow states that were previously established for the application program **224**.

One skilled in the art will recognize that two or more of the previously described API system calls may be combined into a single call or that any one call may be broken down into multiple calls. One skilled in the art will also recognize that the particular names of the API system calls is unimportant. Thus, it is an object of the present invention to cover the foregoing communicating relation between the application program **224** and the flow declaration component **226**, regardless of the particular implementation ultimately chosen.

It should also be understood that any set of values may be inserted in the operation code field **520** of the APD messages provided that each APD message type (e.g., Client Open, Client Accept, Flow Start, etc.) has a different value assigned to it. Furthermore, if a local policy enforcer is unable to handle a particular application program or traffic flow (e.g., insufficient memory or other resources), it preferably responds to the Client Open message with a Client Close message, rather than a Client Accept message.

In the preferred embodiment, the flow declaration component **226** is implemented in software as a series of steps executed at the host/server **222**. Nonetheless, it should be understood that the method may be implemented, either wholly or in part, through one or more computer hardware devices. Additionally, the present invention is preferably utilized only with traffic flows of sufficient length (e.g., greater than 5–10 packets). The application program **224** may be configured not to request bindings or issue API calls for short traffic flows.

It should be understood that some or all of the above described functionality of the local policy enforcer **210** may be located at the host/server **222**. For example, the host/server **222** may include a traffic flow state machine engine **310** that is capable of sending and receiving COPS Request and Decision messages directly to and from the policy server **216**. In this case, the Client Open, Flow Start and Flow Update messages are simply inter-process communications within the host/server **222**, rather than being forwarded across the network. The operating system at the host/server **222** may also include one or more resources that may be utilized to provide traffic management services, such as classifying packets and frames (e.g., loading the DS field **132**, ToS field **122** and/or user_priority field **108**), scheduling packet and frame forwarding from different priority queues, etc.

It should be further understood that the local policy enforcer **210** may make policy or service treatment decisions for traffic flows identified by the flow declaration component **226** without querying the policy server **216**. That is, the local policy enforcer **210** may cache certain policy rules or treatments.

In another aspect of the invention, the application program **224** may request policy decisions in advance of issuing the BeginFlow( ) call **424**. For example, program **224** may only have a small number of application-level parameter bindings. After creating the bindings (using only the application-level parameters) as described above, the program **224** may issue a GetFlowDecision( ) system call to component **226** and, in return, receive a handle, H3. Component **226** issues an Obtain Decision APD message to the local policy enforcer **210** for each binding, including the specified application-level parameters. The local policy enforcer **210** will obtain the appropriate policy rules or service treatments to be applied to these, as yet un-specified, "flows" as described above.

When program **224** is about to begin a flow corresponding to one of these bindings, it may issue a BeginFlow( ) call, including the network and transport layer parameters for the traffic flow and the handle H3 for the corresponding application-level bindings. Component **226** then forwards this information in a Flow Start message **426** to the local policy enforcer **210** as described above. Since the local policy enforcer **210** has already obtained the policy or service treatments to be applied to this flow, it need not query the policy server **216**. Instead, the local policy enforcer **210** simply monitors the traffic from host/server **222** and, when it identifies the specified traffic flow, applies the previously received policy rules or service treatments.

Enhanced RSVP Messaging

In a further aspect of the invention, the flow declaration component **226** may be configured to exchange one or more modified Resource reSerVation Protocol (RSVP) messages with the local policy enforcer **210** in place of the APD messages described above. RSVP is a well-known Internet Control protocol for reserving resources, typically bandwidth, between a sender entity and a receiver entity. RSVP is defined at Request for Comments (RFC) 2205, September 1997, from the Network Working Group of the IETF, and is hereby incorporated by reference in its entirety. The protocol defines two fundamental message types: RSVP path messages (Path) and reservation request messages (Resv). Basically, senders transmit Path message downstream throughout the network to potential receivers offering to supply a given message stream. Receivers, wishing to obtain the proposed message stream, transmit Resv messages that are propagated upstream all the way back to the sender. At each intermediate node in the network, bandwidth resources are reserved to ensure that the receiver will obtain the message stream.

In this embodiment of the present invention, component **226**, rather than generating and forwarding the Flow Start APD message **426** in response to the BeginFlow( ) call **424**, formulates and sends a modified RSVP Path message to the local policy enforcer **210**. FIG. **6** is a block diagram illustrating the preferred format of a modified RSVP Path message **610**. Modified Path message **610** carries the network and transport layer parameters and application-level parameters specified for the anticipated traffic flow. In particular, message **610** preferably includes at least three elements: an RSVP header **612**, a first area **614** (which carries the network and transport layer parameters) and at least one RSVP Policy_Data object **616** (which carries the application-level parameters). As provided in RFC 2205, the RSVP header includes a version field **618**, a flags field **620**, a message type field **622**, an RSVP checksum field **624**, a Send Time To Live (TTL) field **626**, a reserved field **628** and an RSVP length field **630**.

Component **226** preferably loads version field **618**, which corresponds to the version of RSVP, with the appropriate value (e.g., "1"). Flags field **620** is preferably de-asserted as no flags are presently defined. Message type field **622**, which indicates the type of message (e.g., "1" for RSVP Path messages and "2" for RSVP Resv messages) is preferably loaded with the value "1" to indicate that message **610** is a Path message. It should be understood that field **622** may alternatively be loaded with a new value to indicate that message **610** is a modified RSVP Path message. The RSVP Checksum field **624** may be loaded with a computed checksum for message **610**. The Send_TTL_field **626** is preferably loaded with an IP time to live value, and the RSVP length field **630** preferably contains the length of message **610**.

The first area **614** preferably includes an RSVP sender template object **632** and an RSVP session object **634**, each having a plurality of fields. More specifically, the sender template and session objects **632**, **634** each have a length field **638** (loaded with the length of the respective object), a class number field (C-Num) **634** and a class type (C-type) field **642**. For the sender template object **632**, which further includes an IP source address (SA) field **644**, a source port number field **646** and may include one or more un-used fields **648**, the respective C-Num field **640** is preferably loaded with "11" to signify that it is an RSVP sender template object and the respective C-Type field **642** may be loaded with "1" to indicate that fields **644** and **646** carry the IPv4 address and the TCP/UDP port number, respectively, at host/server **222** for the anticipated traffic flow. For the session object **634**, which further includes an IP destination address (DA) field **650**, a transport protocol field **652**, a flags field **654** and a destination port number field **656**, the respective C-Num field **640** is loaded with "1" to signify that it is an RSVP session object and the respective C-Type field **642** may be loaded with "1" to indicate that fields **650** and **656** carry the IPv4 address and the TCP/UDP port number, respectively, for the corresponding process at end station **212** for the anticipated traffic flow. Component **226** may assert flags field **654** if it is capable of policing its own traffic flows.

One skilled in the art will recognize that first area **614** of modified RSVP Path message **610** may be modified in any number of ways, including fewer or additional fields or to carry IPv6 information.

The RSVP Policy_Data object **616** also has a length field **638**, a C-Num field **640** and a C-Type **642** field. In addition RSVP Policy_Data object **616** includes a policy_data object field **658**. The respective length field **638** carries the length of object **616** and the respective C-Num field is loaded with "14" to indicate that field **658** is a policy_data object field. The C-Type field **642** of object **616** is preferably loaded with a new value (e.g., "2") to signify that policy_data object field **658** carries application-level parameters. Furthermore, policy_data object field **658** is loaded by component **226** with the application-level bindings specified by program **224** preferably in the manner as described above with reference to FIG. **5B**.

One skilled in the art will also recognize that the application-level parameters may be carried in multiple RSVP Policy_Data objects **616**.

This modified RSVP path message **610** is preferably handed to the communication facility **228** for forwarding to the local policy enforcer **210** where it is examined. In response, the local policy enforcer **210** and the policy server **216** exchange Request Policy **428** and Policy Decision **430** messages, as described above, in order to obtain the policy rules or service treatments to be applied to the traffic flow identified in the modified RSVP Path message **610**. Local policy enforcer **210** also extracts and stores the network and transport layer parameters from the RSVP Sender Template object **614** in order to identify the particular traffic flow from host/server **222**.

The local policy enforcer **210** may also reply to component **226** with a modified RSVP Resv message rather than the Decision Feedback message **432**. This modified RSVP Resv message preferably includes a header similar to header **612**, but with the message type field **622** loaded with the value "2" to indicate that it is an RSVP Resv messages or with a new value to indicate that it is a modified RSVP Resv message. The modified RSVP Resv message also includes one or more RSVP Policy_Data objects similar to object **616**. In this case, however, object **616** carries the decision bindings for the anticipated traffic flow as described above. Component **226** may extract these decision bindings in order to provide feedback to application **224**.

As shown, component **226** utilizes a modified RSVP path message **610** to identify network and transport layer parameters and application-level parameters to the local policy enforcer **210**. The modified RSVP Path message **610**, moreover, is preferably not forwarded by the local policy enforcer **210**, unlike conventional RSVP Path and Resv messages which are propagated all the way between the sender and receiver entities.

It should be understood that the local policy enforcer **210** is preferably in close proximity to host/server **222** so that the classification of packets or frames from the anticipated traffic flow occurs early in their journey through the network **200**. It should also be understood that the traffic flow from end station **212** to host/server **222** may similarly be identified and appropriate policy rules or service treatments applied thereto. It should be further understood that the flow declaration component **226** is configured to handle and separately identify multiple traffic flows from multiple application programs executing at the host/server **222** so that the appropriate policy rules or service treatments may be individually applied to each such traffic flow through the network **200**. For example, program **224** may be simultaneously sending a print transaction t the print server **214**.

The foregoing description has been directed to specific embodiments of the invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. For example, other client-server communications protocols, besides COPS, may be utilized by the policy server and the local policy enforcer. In addition, the present invention may also be utilized with other network layer protocols, such as IPv6, whose addresses are 128 bits long. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A network entity configured to communicate with a local policy enforcer through a computer network, the network entity having at least one application program executing thereon for generating a traffic flow for transmission to a second network entity through the network, the network entity comprising:

a flow declaration component in communicating relation with the at least one application program for receiving one or more network and transport layer parameters and one or more application-level parameters identifying the traffic flow, the flow declaration component comprising:

a memory for storing a traffic flow data structure corresponding to the traffic flow, the traffic flow data structure storing the one or more network and transport layer and one or more application-level parameters identified by the at least one application program, and

a message generator for formulating and transmitting one or more messages to the local policy enforcer, at least one message including information from the traffic flow data structure,

whereby, in response to the at least one message from the flow declaration component, a respective service treatment is declared for the traffic flow from the at least one application program.

2. The network entity of claim **1** wherein the at least one application program communicates with the flow declaration component through one or more Application Programming Interface (API) system calls.

3. The network entity of claim **2** wherein the flow declaration component, in response to one or more API system calls from the at least one application program, associates the traffic flow data structure with the at least one application program.

4. The network entity of claim **3** wherein the flow declaration component, in response to one or more API system calls from the at least one application program that specify one or more network and transport layer parameters for the traffic flow, fills the traffic flow data structure with the specified one or more network and transport layer parameters.

5. The network entity of claim **4** wherein the flow declaration component, in response to one or more API system calls from the at least one application program specifying a start of the traffic flow, generates a message having the specified one or more network and transport layer parameters for transmission to the local policy enforcer.

6. The network entity of claim **3** wherein the flow declaration component, in response to one or more API system calls from the at least one application program that specify one or more application-level parameters for the traffic flow, fills the traffic flow data structure with the specified one or more application-level parameters.

7. The network entity of claim **6** wherein the flow declaration component, in response to one or more API system calls from the at least one application program specifying a start of the traffic flow, generates a message having the specified one or more application-level parameters for transmission to the local policy enforcer.

8. The network entity of claim **4** wherein the flow declaration component, in response to one or more API system calls from the at least one application program that specify one or more application-level parameters for the traffic flow, fills the traffic flow data structure with the specified one or more application-level parameters.

9. The network entity of claim **8** wherein the flow declaration component, in response to one or more API system calls from the at least one application program specifying a start of the traffic flow, generates a first message having the specified one or more network and transport layer parameters and the specified one or more application-level parameters for transmission to the local policy enforcer.

10. The network entity of claim **9** wherein the first message formulated by the message generator is a Resource reSerVation Protocol (RSVP) Path message that includes:

a first area carrying the specified one or more network and transport layer parameters; and

one or more policy data objects carrying the specified one or more application-level parameters.

11. The network entity of claim **10** wherein the first area of the Resource reSerVation Protocol (RSVP) Path message includes a session object and a sender template object.

12. A network entity configured to communicate with a policy server through a computer network, the network entity having at least one application program executing thereon for generating a traffic flow for transmission to a second network entity through the network, the network entity comprising:

a flow declaration component in communicating relation with the at least one application program for receiving one or more network and transport layer parameters and one or more application-level parameters identifying the traffic flow, the flow declaration component having a memory for storing a traffic flow data structure corresponding to the traffic flow, the traffic flow data structure storing the one or more network and transport layer and the one or more application-level parameters identified by the at least one application program; and

a traffic flow state machine engine in communication with the flow declaration component, the traffic flow state machine engine including a communication engine for formulating and transmitting one or more messages to the policy server carrying information from the traffic flow data structure and for receiving one or more policy decision rules from the policy server to be applied to the traffic flow.

13. The network entity of claim 12 wherein the flow declaration component, in response to one or more Application Programming Interface (API) system calls from the at least one application program that specify one or more network and transport layer parameters and one or more application-level parameters for the traffic flow, fills the traffic flow data structure with the specified one or more network and transport layer parameters and the one or more application-level parameters.

14. The network entity of claim 13 wherein the communication engine at the traffic flow state machine engine, in response to one or more API system calls from the at least one application program specifying a start of the traffic flow, generates a first message having the specified one or more network and transport layer parameters and the one or more specified application-level parameters for transmission to the policy server.

15. In a computer network having a first network entity, a local policy enforcer, a policy server and a second network entity, the first network entity having a flow declaration component and at least one application program that are in communicating relation, the at least one application program configured to generate a traffic flow for transmission through the network to the second network entity, a method for obtaining and applying policy rules to the traffic flow comprising the steps of:

specifying one or more network and transport layer parameters for the traffic flow to the flow declaration component;

specifying one or more application-level parameters that describes an aspect of the traffic flow to the flow declaration component;

forwarding at least one message carrying the specified network and transport layer parameters and the specified application-level parameters from the flow declaration component to the local policy enforcer;

at the local policy enforcer, requesting a policy rule decision for application to the traffic flow from the policy server based on the specified network and transport layer parameters and the specified application-level parameters; and

at the local policy enforcer, applying the policy rule decision to the traffic flow as it moves through the network.

16. The method of claim 15 wherein the at least one application program specifies the one or more network and transport layer parameters and the one or more application-level parameters to the flow declaration component through one or more Application Programming Interface (API) system calls.

17. The method of claim 16 further wherein the traffic flow has a start and the method further comprises the step of notifying the flow declaration component of the start of the traffic flow and further wherein the flow declaration component, in response, forwards the at least one message to the local policy enforcer.

18. The method of claim 17 wherein the at least one message forwarded by the flow declaration component is a Resource reSerVation Protocol (RSVP) Path message that includes:

a first area carrying the specified one or more network and transport layer parameters; and

one or more policy data objects carrying the specified one or more application-level parameters.

19. The method of claim 18 wherein the first area of the Resource reSerVation Protocol (RSVP) Path message includes a session object and a sender template object.

20. A computer readable medium containing executable program instructions for declaring a service treatment for a plurality of network messages issued by an application program running on a network entity connected to a computer network, the network messages corresponding to a specific traffic flow, the computer network including a policy enforcer and a policy server and defining transport and network communication layers, the executable program instructions comprising program instructions for:

receiving from the application program a plurality of network and transport layer parameters corresponding to the traffic flow;

receiving from the application program one or more application-level parameters corresponding to the traffic flow;

loading the received network and transport layer parameters and the application-level parameters into one or more flow start messages; and

sending the one or more flow start messages to the policy enforcer, wherein, in response to the one or more flow start message a service treatment is obtained for and applied to the traffic flow from the application program.

21. The computer readable medium of claim 20 further comprising program instructions for receiving a notification from the application program indicating that the program is ready to begin sending the network messages of the traffic flow.

22. The computer readable medium of claim 21 further comprising program instructions for:

receiving a notification from the application program indicating that the program has completed its sending of messages corresponding to the traffic flow, and

issuing a flow end message to the policy enforcer signaling the end of the traffic flow.

23. The computer readable medium of claim 22 wherein the application-level parameters specify one or more of the following characteristics: the size of a file being transmitted, a video segment name, a video segment viewer, a user name, a user department, an application module identifier, a transaction type, a transaction name, an application state, a calling party, a called party, a compression method, a service level, a uniform resource locator (URL) and a mime type.

24. The computer readable medium of claim 23 further comprising program instructions for loading the received network and transport layer parameters and the received application level-parameters into a traffic flow data structure associated with the application program.

25. The computer readable medium of claim 20 wherein the one or more flow start messages contain one or more policy bindings, the policy bindings representing encoded versions of the network and transport layer parameters received from the application program.

26. The computer readable medium of claim 25 wherein the policy bindings further represent encoded versions of the application-level parameters received from the application program.

27. The computer readable medium of claim 26 wherein each policy binding includes a policy identifier (PID) element and an encoded policy instance element.

28. The computer readable medium of claim 27 wherein the PID is used to specify a type of class of the network and transport layer parameters and/or the application-level parameters.

29. The computer readable medium of claim 27 wherein the PID elements comply with the COPS Usage for Differentiated Services specification standard.

30. The computer readable medium of claim 26 further comprising program instructions for translating the application-level parameters into a machine independent format.

31. The computer readable medium of claim 30 wherein the machine independent format is Abstract Syntax Notation One (ASN.1).

32. The computer readable medium of claim 20 further comprising programming instructions for providing the service treatments to the application program through a call-back function.

33. The computer readable medium of claim 20 further comprising programming instructions for sending one or more client open messages to the policy enforcer in order to open a communication session with the policy enforcer, the client open messages carrying a keep alive timer value.

34. The computer readable medium of claim 33 further comprising programming instructions for receiving one or more client accept messages from the policy enforcer, the Client Accept messages carrying a keep alive timer value.

35. The computer readable medium of claim 34 further comprising programming instructions for issuing one or more keep alive messages to the policy enforcer while the application program continues to send network messages corresponding to the traffic flow, the keep alive message sent substantially in accordance with the keep alive timer valve received from the policy enforcer.

36. The computer readable medium of claim 33 wherein a separate communication session is opened with the policy enforcer for each application program for which a service treatment is to be declared.

37. The computer readable medium of claim 21 further comprising program instructions for receiving from the application program a change in the application-level parameters for the network messages corresponding to the traffic flow.

38. The computer readable medium of claim 37 further comprising program instructions for:

receiving a notification from the application program indicating that the program is ready to begin sending the network messages corresponding to the changed application-level parameters; and

issuing one or more flow update messages to the policy enforcer, the flow update messages containing the changed application-level parameters.

39. The computer readable medium of claim 38 wherein a new service treatment is obtained for and applied to the network messages corresponding to the changed application-level parameters from the application program.

40. The computer readable medium of claim 24 wherein the service treatment is obtained in response to the policy enforcer sending one or more request policy messages to the policy server.

41. The computer readable medium of claim 40 wherein the request policy messages comply in substantial part with the Common Open Policy Service (COPS) Protocol.

42. The computer readable medium of claim 41 wherein the policy server, in response to the request policy messages, issues one or more policy decision messages to the policy enforcer, the policy decision messages containing the service treatment for the traffic flow from the application program.

43. The computer readable medium of claim 42 wherein the policy enforcer establishes a flow state for the traffic flow from the application program, the flow state including the declared network and transport layer parameters and the service treatment returned by the policy server.

44. The computer readable medium of claim 43 wherein the policy enforcer

compares messages originated by the application program with the declared network and transport layer parameters, and

applies the service treatment to messages matching the network and transport layer parameters.

45. The computer readable medium of claim 44 wherein the policy enforcer, in applying the service treatment, performs one or more of:

setting a Differentiated Services (DS) codepoint field of matching network messages from the application program,

setting a Type of Service (ToS) field of matching network messages from the application program, and

setting a user_priority field of matching messages from the application program.

46. The computer readable medium of claim 43 further comprising program instructions for discarding the contents of the traffic flow data structure in response to receiving the notification from the application program that the program has completed its sending of messages.

47. The computer readable medium of claim 46 wherein the policy enforcer, in response to the flow end message, erases the traffic flow state established for the traffic flow from the application program.

* * * * *